INTERNATIONAL STANDARD

ISO/IEC 9075-4

First edition
1996-12-15

Information technology — Database languages — SQL —

Part 4:
Persistent Stored Modules (SQL/PSM)

Technologies de l'information — Langages de base de données — SQL — Partie 4: Modules mémorisés persistants SQL (SQL/PSM)

Reference number
ISO/IEC 9075-4:1996(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 9075-4, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC21, *Open systems interconnection, data management, and open distributed processing*.

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

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

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

Parts 1, 2, and 5 are currently published as ISO/IEC 9075:1992.

Part 3 is currently published as ISO/IEC 9075-3:1995.

Annexes A, B, C, and D 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 stored modules.

5) Clause 5, "Lexical elements", defines a number of lexical elements used in the definition of persistent stored modules.

6) Clause 6, "Scalar expressions", defines a number of scalar expressions used in the definition of persistent stored modules.

7) Clause 7, "Query expressions", defines the elements of the language that produce rows and tables of data as used in persistent stored modules.

8) Clause 8, "Data assignment rules and routine determination", defines the data assignment rules used in the definition of persistent stored modules.

9) Clause 9, "Additional common elements", defines additional common elements used in the definition of persistent stored modules.

10) Clause 10, "Schema definition and manipulation", defines the schema definition and manipulation statements associated with the definition of persistent stored modules.

11) Clause 11, "Modules", defines the facilities for using persistent stored modules.

12) Clause 12, "Data manipulation", defines data manipulation operations associated with persistent stored modules.

13) Clause 13, "Control statements", defines the control statements used with persistent stored modules.

14) Clause 14, "Transaction management", defines the SQL-transaction management statements associated with persistent stored modules.

15) Clause 15, "Session management", defines the SQL-session management statements associated with persistent stored modules.

16) Clause 16, "Dynamic SQL", defines the facilities for executing SQL-statements dynamically in the context of persistent stored modules.

17) Clause 17, "Diagnostics management", defines enhancements to the facilities used with persistent stored modules.

18) Clause 18, "Embedded SQL", defines host language embeddings related to persistent stored modules.

19) Clause 19, "Information Schema and Definition Schema", defines the Information and Definition Schema objects associated with persistent stored modules.

20) Clause 20, "Status codes", defines SQLSTATE values related to persistent stored modules.

21) Clause 21, "Conformance", defines the criteria for conformance to this part of ISO/IEC 9075.

22) Annex A, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this part of the standard states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementer shall provide in each case.

23) Annex B, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of this part of the standard states that the syntax or meaning or effect on the database is partly or wholly implementation-dependent.

24) Annex C, "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.

25) Annex D, "Incompatibilities with ISO/IEC 9075:1992", is an informative Annex. It lists the incompatibilities between this edition of ISO/IEC 9075 and ISO/IEC 9075:1992.

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 4:
Persistent Stored Modules (SQL/PSM)

1 Scope

This part of International Standard ISO/IEC 9075 specifies the syntax and semantics of a database language for declaring and maintaining persistent database language routines either in SQL-server modules or as standalone schema-level routines, and invoking them from programs written in a standard programming language.

The database language for <procedure>s and <SQL-invoked routine>s includes:

— The specification of statements to direct the flow of control.

— The assignment of the result of expressions to variables and parameters.

— The specification of condition handlers that allow SQL-invoked routines to deal with various conditions that arise during their execution.

— The specification of statements to signal and resignal conditions.

— The ability to set an SQL-path for controlling the determination of the subject routine to be invoked.

— The declaration of local cursors.

— The declaration of local variables.

It also includes the definition of the Information Schema tables that contain schema information pertaining to SQL-server modules and SQL-invoked routines.

NOTE 1 – The context for ISO/IEC 9075 is described by the Reference Model of Data Management (ISO/IEC 10032:1993).

2 Persistent Stored Modules (SQL/PSM)

2 Normative references

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 1539:1991, *Information technology — Programming languages — FORTRAN.*

ISO 1989:1985, *Programming languages — COBOL.*

ISO 6160:1979, *Programming languages — PL/I.*

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

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

NOTE 2 – ISO 8652:1987 has been superseded by a new edition (ISO/IEC 8652:1995). However, when this part of ISO/IEC 9075 was under development, the previous edition was valid and this part of ISO/IEC 9075 is therefore based on that edition, which is listed below.

ISO 8652:1987, *Programming languages — Ada.*

ISO/IEC 9075:1992, *Information technology — Database languages — SQL.*

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

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 9075:1992 and the following definitions apply.

a) **external routine**: An SQL-invoked routine whose routine body is an external body reference that identifies a program written in a standard programming language other than SQL.

b) **signature (of an SQL-invoked routine)**: The name of the SQL-invoked routine, the position and data types of each of its SQL parameters, and an indication of whether it is an SQL-invoked function or an SQL-invoked procedure.

c) **SQL routine**: An SQL-invoked routine whose routine body is written in SQL.

d) **SQL-invoked routine**: A routine that is allowed to be invoked only from within SQL.

3.2 Notations

Insert this paragraph All notations in ISO/IEC 9075:1992 apply to this part of ISO/IEC 9075.

Insert this paragraph The syntax notation used in this part of ISO/IEC 9075 is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form").

Insert this paragraph This version of BNF is fully described in ISO/IEC 9075:1992.

3.3 Conventions

Insert this paragraph Except as otherwise specified in this part of ISO/IEC 9075, the conventions used in this part of ISO/IEC 9075 are identical to those described in part 1 of ISO/IEC 9075.

3.3.1 Use of terms

3.3.1.1 Exceptions

Modified paragraph The phrase "an exception condition is raised:", followed by the name of a condition, is used in General Rules and elsewhere to indicate that:

— The execution of a statement is unsuccessful.

— The application of General Rules, other than those of Subclause 12.3, "<procedure>", in ISO/IEC 9075:1992, Subclause 9.1, "<routine invocation>", Subclause 11.4, "<SQL procedure statement>", Subclause 20.1, "<direct SQL statement>", in ISO/IEC 9075:1992, Subclause 13.3, "<compound statement>", and Subclause 13.4, "<handler declaration>", may be terminated.

— Diagnostic information is to be made available.

— Execution of the statement is to have no effect on SQL-data or schemas.

Insert this paragraph The effect on <target specification>s and SQL descriptor areas of an SQL-statement that terminates with an exception condition, unless explicitly defined by this International Standard, is implementation-dependent.

Insert this paragraph The phrase "*C* is re-raised by *S*" is used in General Rules and elsewhere to indicate that *C*, a condition raised by an SQL-statement executed during execution of *S*, is raised again by *S*.

3.3.1.2 Syntactic containment

Insert this paragraph If <A> contains a <table name> that identifies a view that is defined by a <view definition> *V*, then <A> is said to *generally contain* the <query expression> contained in *V*. If <A> contains a <routine invocation> *RI*, then <A> is said to *generally contain* the <routine body>s of all <SQL-invoked routine>s that are included in the set of subject routines of *RI*. If <A> contains <B>, then <A> generally contains <B>. If <A> generally contains <B> and <B> generally contains <C>, then <A> generally contains <C>.

3.3.1.3 Rule evaluation order

Insert this paragraph An invocation of an SQL-invoked function is *inessential* if the SQL-function is deterministic and does not possibly modify SQL-data; otherwise, it is implementation-defined whether or not it is inessential.

3.3.2 Relationships to ISO/IEC 9075:1992

This part of ISO/IEC 9075 depends on ISO/IEC 9075:1992 and its Technical Corrigenda. This part of ISO/IEC 9075 is to be used as though it were merged with the text of ISO/IEC 9075:1992. This Subclause describes the conventions used to specify the merger. The merger described also accounts for the Technical Corrigenda that have been published to correct ISO/IEC 9075:1992. This accommodation is typically indicated by the presence of a phrase like "in the Technical Corrigenda" or "in the TC".

3.3.2.1 New and modified Clauses, Subclauses, and Annexes

Clauses, Subclauses, and Annexes (other than Clause 1, "Scope" and Clause 2, "Normative references") in this part of ISO/IEC 9075 that have names identical to Clauses, Subclauses, or Annexes in ISO/IEC 9075:1992 supplement the Clause, Subclause, or Annex, respectively, in ISO/IEC 9075:1992, typically by replacing paragraphs, Format items, or Rules or by providing new paragraphs, Format items, or Rules. However, Clauses, Subclauses, and Annexes in this part of ISO/IEC 9075 that have names identical to Clauses, Subclauses, and Annexes in ISO/IEC 9075:1992 do not necessarily have the same *number* or *letter* as the corresponding Clause, Subclause, or Annex in ISO/IEC 9075:1992. Any differences in Clause or Subclause number or in Annex letter is not significant. Table 1, "Clause, Subclause, and Table relationships", identifies the relationships between Clauses, Subclauses, and Annexes in this part of ISO/IEC 9075 and the corresponding Clauses, Subclauses, and Annexes in ISO/IEC 9075:1992.

Clauses, Subclauses, and Annexes in this part of ISO/IEC 9075 that have names that are not identical to Clauses, Subclauses, or Annexes in ISO/IEC 9075:1992 provide language specification particular to this part of ISO/IEC 9075. Subclauses that are subsidiary to ("contained in") Clauses or Subclauses identified as new are inherently new and are not further identified.

The Clauses, Subclauses, and Annexes in this part of ISO/IEC 9075 appear in the order in which they are intended to appear in the merged document. Absent other explicit instructions regarding its placement, any new Clause, Subclause, or Annex is to be positioned as follows: Locate the prior Clause, Subclause, or Annex in this part of ISO/IEC 9075 whose name is identical to the name of a corresponding Clause, Subclause, or Annex that appears in ISO/IEC 9075:1992. The new Clause, Subclause, or Annex shall immediately follow that Clause, Subclause, or Annex. If there are multiple new Clauses, Subclauses, or Annexes with no intervening Clause, Subclause, or Annex that modifies an existing Clause, Subclause, or Annex, then those new Clauses, Subclauses, or Annexes appear in order, following the prior Clause, Subclause, or Annex whose name was matched.

3.3.2.2 New and modified Format items

In modified Subclauses, Format items that define a BNF nonterminal symbol (that is, the BNF nonterminal symbol appears on the left-hand side of the `::=` mark) sometimes modify a Format item whose definition appears in ISO/IEC 9075:1992, sometimes replace a Format item whose definition appears in ISO/IEC 9075:1992, and sometimes define a new Format item that does not have a definition at all in ISO/IEC 9075:1992. Those Format items in this part of ISO/IEC 9075 that modify a Format item whose definition appears in ISO/IEC 9075:1992, are identified by the existence of a "Format comment" such as:

```
<modified item> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <new alternative>
```

By contrast, Format items that completely replace Format items in ISO/IEC 9075:1992 have BNF nonterminal symbols identical to BNF nonterminal symbols of Format items in ISO/IEC 9075:1992, but do not state that they include any alternatives from ISO/IEC 9075:1992.

New Format items that have no correspondence to any Format item in ISO/IEC 9075:1992 are not specially identified in this part of ISO/IEC 9075.

In new Subclauses, all Format items are also new and require no specific marking.

3.3.2.3 New and modified paragraphs and rules

In modified Subclauses, each paragraph or Rule is marked to indicate whether it is a modification of a paragraph or Rule in ISO/IEC 9075:1992, or is a new paragraph or Rule in this part of ISO/IEC 9075.

Modifications of paragraphs or Rules in ISO/IEC 9075:1992, are identified by the inclusion of an indication such as Replace the 5th paragraph, meaning that the fifth paragraph of the corresponding Subclause in ISO/IEC 9075:1992, is to be replaced by the paragraph to which the indication is attached, or Replace SR6)b)ii), meaning that Syntax Rule 6)b)ii) of the corresponding Subclause in ISO/IEC 9075:1992, is to be replaced by the rule to which the indication is attached. In some cases, an indication such as Augment SR3) is used. When this appears, it means that the referenced Rule is extended or enhanced by the Rule to which the indication is attached. In most instances, the augmentation is the addition of a new alternative meant to support new syntax.

New paragraphs or Rules in this part of ISO/IEC 9075 are indicated by the inclusion of an indication such as Insert before 2nd paragraph, meaning that the paragraph to which the indication is attached is to be read as though it were inserted preceding the second paragraph of the corresponding Subclause in ISO/IEC 9075:1992. Insert before GR4) should be interpreted to mean that the rule to which the indication is attached is to be read as though it were inserted preceding General Rule 4) of the corresponding Subclause in ISO/IEC 9075:1992. When an indication does not indicate a specific insertion point, such as Insert this paragraph or Insert this GR, then it may be read as implicitly specifying that the new text is to be appended at the end of the appropriate section (the General Rules, for example) of the corresponding Subclause in ISO/IEC 9075:1992.

In such instructions, "SR" is used to mean "Syntax Rule", "AR" is used to mean "Access Rule", and "GR" is used to mean "General Rule". "Desc." is used to mean "Description" and "Func." is used to mean "Function".

All paragraphs, Format items, and Rules in new Clauses or Subclauses are also new and therefore do not require further identification.

3.3.2.4 New and modified tables

Similarly, tables in this part of ISO/IEC 9075 that have names that are identical to tables in ISO/IEC 9075:1992 supplement the table in ISO/IEC 9075:1992, typically by adding or replacing one or more table entries. Tables in this part of ISO/IEC 9075 that have names that are not identical to the names of tables in ISO/IEC 9075:1992 are new tables specified by this part of ISO/IEC 9075. Table 1, "Clause, Subclause, and Table relationships", identifies the relationships between tables in this part of ISO/IEC 9075 and the corresponding tables in ISO/IEC 9075:1992.

The rows in modified tables are generally new rows to be effectively inserted into the corresponding table in ISO/IEC 9075:1992, though in rare cases rows already in tables in ISO/IEC 9075:1992 are effectively replaced by rows in the table in this part of ISO/IEC 9075. It is always obvious when such an effective replacement is required, based on equality of values in the first column of the table.

3.3.2.5 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 ISO/IEC 9075:1992
Clause 1, "Scope"	Clause 1, "Scope"
Clause 2, "Normative references"	Clause 2, "Normative references"
Clause 3, "Definitions, notations, and conventions"	Clause 3, "Definitions, notations, and conventions"
Subclause 3.1, "Definitions"	Subclause 3.1, "Definitions"
Subclause 3.2, "Notations"	Subclause 3.2, "Notation"
Subclause 3.3, "Conventions"	Subclause 3.3, "Conventions"
Subclause 3.3.1, "Use of terms"	Subclause 3.3.4, "Use of terms"
Subclause 3.3.1.1, "Exceptions"	Subclause 3.3.4.1, "Exceptions"

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 ISO/IEC 9075:1992
Subclause 3.3.1.2, "Syntactic containment"	Subclause 3.3.4.2, "Syntactic containment"
Subclause 3.3.1.3, "Rule evaluation order"	Subclause 3.3.4.4, "Rule evaluation order"
Subclause 3.3.2, "Relationships to ISO/IEC 9075:1992"	*(none)*
Subclause 3.3.2.1, "New and modified Clauses, Subclauses, and Annexes"	*(none)*
Subclause 3.3.2.2, "New and modified Format items"	*(none)*
Subclause 3.3.2.3, "New and modified paragraphs and rules"	*(none)*
Subclause 3.3.2.4, "New and modified tables"	*(none)*
Subclause 3.3.2.5, "Clause, Subclause, and Table relationships"	*(none)*
Subclause 3.4, "Object identifier for Database Language SQL"	Subclause 3.4, "Object identifier for Database Language SQL"
Clause 4, "Concepts"	Clause 4, "Concepts"
Subclause 4.1, "SQL-server Modules"	*(none)*
Subclause 4.2, "SQL-invoked routines"	*(none)*
Subclause 4.3, "SQL-paths"	*(none)*
Subclause 4.3.1, "Type conversions and mixing of data types"	Subclause 4.6, "Type conversions and mixing of data types"
Subclause 4.4, "Tables"	Subclause 4.9, "Tables"
Subclause 4.5, "Integrity constraints"	Subclause 4.10, "Integrity constraints"
Subclause 4.6, "SQL-schemas"	Subclause 4.11, "SQL-schemas"
Subclause 4.7, "Parameters"	Subclause 4.18, "Parameters"
Subclause 4.7.1, "Status parameters"	Subclause 4.18.1, "Status parameters"
Subclause 4.8, "Diagnostics area"	Subclause 4.19, "Diagnostics area"
Subclause 4.9, "Cursors"	Subclause 4.21, "Cursors"
Subclause 4.10, "Condition handling"	*(none)*
Subclause 4.11, "SQL-statements"	Subclause 4.22, "SQL-statements"
Subclause 4.11.1, "SQL-statements classified by function"	Subclause 4.22.2, "SQL-statements classified by function"
Subclause 4.11.2, "Embeddable SQL-statements"	Subclause 4.22.3, "Embeddable SQL-statements"
Subclause 4.11.3, "Preparable and immediately executable SQL-statements"	Subclause 4.22.4, "Preparable and immediately executable SQL-statements"
Subclause 4.11.4, "Directly executable SQL-statements"	Subclause 4.22.5, "Directly executable SQL-statements"

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 ISO/IEC 9075:1992
Subclause 4.11.5, "SQL-statements and transaction states"	Subclause 4.22.6, "SQL-statements and transaction states"
Subclause 4.11.6, "Compound statements"	*(none)*
Subclause 4.11.7, "SQL-statement atomicity"	*(none)*
Subclause 4.12, "Privileges"	Subclause 4.26, "Privileges"
Subclause 4.13, "SQL-sessions"	Subclause 4.30, "SQL-sessions"
Clause 5, "Lexical elements"	Clause 5, "Lexical elements"
Subclause 5.1, "<token> and <separator>"	Subclause 5.2, "<token> and <separator>"
Subclause 5.2, "Names and identifiers"	Subclause 5.4, "Names and identifiers"
Clause 6, "Scalar expressions"	Clause 6, "Scalar expressions"
Subclause 6.1, "<value specification> and <target specification>"	Subclause 6.2, "<value specification> and <target specification>"
Subclause 6.2, "<column reference>"	Subclause 6.4, "<column reference>"
Subclause 6.3, "<item reference>"	*(none)*
Subclause 6.4, "<datetime value function>"	Subclause 6.8, "<datetime value function>"
Subclause 6.5, "<case expression>"	Subclause 6.9, "<case expression>"
Subclause 6.6, "<value expression>"	Subclause 6.11, "<value expression>"
Clause 7, "Query expressions"	Clause 7, "Query expressions"
Subclause 7.1, "<table value constructor>"	Subclause 7.2, "<table value constructor>"
Subclause 7.2, "<table expression>"	Subclause 7.3, "<table expression>"
Subclause 7.3, "<query specification>"	Subclause 7.9, "<query specification>"
Subclause 7.4, "<query expression>"	Subclause 7.10, "<query expression>"
Subclause 7.5, "<scalar subquery>, <row subquery>, and <table subquery>"	Subclause 7.11, "<scalar subquery>, <row subquery>, and <table subquery>"
Clause 8, "Data assignment rules and routine determination"	Clause 9, "Data assignment rules"
Subclause 8.1, "Subject routine determination"	*(none)*
Subclause 8.2, "Type precedence list determination"	*(none)*
Clause 9, "Additional common elements"	Clause 10, "Additional common elements"
Subclause 9.1, "<routine invocation>"	*(none)*
Subclause 9.2, "<privileges>"	Subclause 10.3, "<privileges>"
Subclause 9.3, "<specific routine designator>"	*(none)*
Subclause 9.4, "<sqlstate value>"	*(none)*
Subclause 9.5, "<language clause>"	Subclause 10.2, "<language clause>"

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 ISO/IEC 9075:1992
Subclause 9.6, "<path specification>"	*(none)*
Clause 10, "Schema definition and manipulation"	Clause 11, "Schema definition and manipulation"
Subclause 10.1, "<schema definition>"	Subclause 11.1, "<schema definition>"
Subclause 10.2, "<drop schema statement>"	Subclause 11.2, "<drop schema statement>"
Subclause 10.3, "<default clause>"	Subclause 11.5, "<default clause>"
Subclause 10.4, "<check constraint definition>"	Subclause 11.9, "<check constraint definition>"
Subclause 10.5, "<drop column definition>"	Subclause 11.15, "<drop column definition>"
Subclause 10.6, "<drop table constraint definition>"	Subclause 11.17, "<drop table constraint definition>"
Subclause 10.7, "<drop table statement>"	Subclause 11.18, "<drop table statement>"
Subclause 10.8, "<view definition>"	Subclause 11.19, "<view definition>"
Subclause 10.9, "<drop view statement>"	Subclause 11.20, "<drop view statement>"
Subclause 10.10, "<drop domain statement>"	Subclause 11.27, "<drop domain statement>"
Subclause 10.11, "<drop character set statement>"	Subclause 11.29, "<drop character set statement>"
Subclause 10.12, "<drop collation statement>"	Subclause 11.31, "<drop collation statement>"
Subclause 10.13, "<drop translation statement>"	Subclause 11.33, "<drop translation statement>"
Subclause 10.14, "<assertion definition>"	Subclause 11.34, "<assertion definition>"
Subclause 10.15, "<drop assertion statement>"	Subclause 11.35, "<drop assertion statement>"
Subclause 10.16, "<SQL-server module definition>"	*(none)*
Subclause 10.17, "<drop module statement>"	*(none)*
Subclause 10.18, "<SQL-invoked routine>"	*(none)*
Subclause 10.19, "<drop routine statement>"	*(none)*
Subclause 10.20, "<grant statement>"	Subclause 11.36, "<grant statement>"
Subclause 10.21, "<revoke statement>"	Subclause 11.37, "<revoke statement>"
Clause 11, "Modules"	Clause 12, "Module"
Subclause 11.1, "<module>"	Subclause 12.1, "<module>"
Subclause 11.2, "<procedure>"	Subclause 12.3, "<procedure>"
Subclause 11.3, "Calls to a <procedure>"	Subclause 12.4, "Calls to a <procedure>"
Subclause 11.4, "<SQL procedure statement>"	Subclause 12.5, "<SQL procedure statement>"
Subclause 11.5, "Data type correspondences"	*(none)*
Clause 12, "Data manipulation"	Clause 13, "Data manipulation"
Subclause 12.1, "<declare cursor>"	Subclause 13.1, "<declare cursor>"

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 ISO/IEC 9075:1992
Subclause 12.2, “<open statement>”	Subclause 13.2, "<open statement>"
Subclause 12.3, “<fetch statement>”	Subclause 13.3, "<fetch statement>"
Subclause 12.4, “<close statement>”	Subclause 13.4, "<close statement>"
Subclause 12.5, “<select statement: single row>”	Subclause 13.5, "<select statement: single row>"
Subclause 12.6, “<delete statement: positioned>”	Subclause 13.6, "<delete statement: positioned>"
Subclause 12.7, “<delete statement: searched>”	Subclause 13.7, "<delete statement: searched>"
Subclause 12.8, “<update statement: positioned>”	Subclause 13.9, "<update statement: positioned>"
Subclause 12.9, “<update statement: searched>”	Subclause 13.10, "<update statement: searched>"
Subclause 12.10, “<temporary table declaration>”	Subclause 13.11, "<temporary table declaration>"
Clause 13, “Control statements”	*(none)*
Subclause 13.1, “<call statement>”	*(none)*
Subclause 13.2, “<return statement>”	*(none)*
Subclause 13.3, “<compound statement>”	*(none)*
Subclause 13.4, “<handler declaration>”	*(none)*
Subclause 13.5, “<condition declaration>”	*(none)*
Subclause 13.6, “<SQL variable declaration>”	*(none)*
Subclause 13.7, “<assignment statement>”	*(none)*
Subclause 13.8, “<case statement>”	*(none)*
Subclause 13.9, “<if statement>”	*(none)*
Subclause 13.10, “<leave statement>”	*(none)*
Subclause 13.11, “<loop statement>”	*(none)*
Subclause 13.12, “<while statement>”	*(none)*
Subclause 13.13, “<repeat statement>”	*(none)*
Subclause 13.14, “<for statement>”	*(none)*
Clause 14, “Transaction management”	Clause 14, "Transaction management"
Subclause 14.1, “<commit statement>”	Clause 14.3, "<commit statement>"
Subclause 14.2, “<rollback statement>”	Clause 14.4, "<rollback statement>"
Clause 15, “Session management”	Clause 16, "Session management"
Subclause 15.1, “<set path statement>”	*(none)*
Clause 16, “Dynamic SQL”	Clause 17, "Dynamic SQL"
Subclause 16.1, “Description of SQL item descriptor areas”	Subclause 17.1, "Description of SQL item descriptor areas"
Subclause 16.2, “<prepare statement>”	Subclause 17.6, "<prepare statement>"

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 ISO/IEC 9075:1992
Subclause 16.3, “<using clause>”	Subclause 17.9, "<using clause>"
Clause 17, “Diagnostics management”	Clause 18, "Diagnostics management"
Subclause 17.1, “<get diagnostics statement>”	Subclause 18.1, "<get diagnostics statement>"
Subclause 17.2, “<signal statement>”	*(none)*
Subclause 17.3, “<resignal statement>”	*(none)*
Clause 18, “Embedded SQL”	Clause 19, "Embedded SQL"
Subclause 18.1, “<embedded SQL host program>”	Subclause 19.1, "<embedded SQL host program>"
Clause 19, “Information Schema and Definition Schema”	Clause 21, "Information Schema and Definition Schema"
Subclause 19.1, “Information Schema”	Subclause 21.2, "Information Schema"
Subclause 19.1.1, “SCHEMATA view”	Subclause 21.2.4, "SCHEMATA view"
Subclause 19.1.2, “DOMAINS view”	Subclause 21.2.5, "DOMAINS view"
Subclause 19.1.3, “COLUMNS view”	Subclause 21.2.9, "COLUMNS view"
Subclause 19.1.4, “MODULES view”	*(none)*
Subclause 19.1.5, “ROUTINES view”	*(none)*
Subclause 19.1.6, “PARAMETERS view”	*(none)*
Subclause 19.1.7, “MODULE_TABLE_USAGE view”	*(none)*
Subclause 19.1.8, “MODULE_COLUMN_USAGE view”	*(none)*
Subclause 19.1.9, “MODULE_PRIVILEGES view”	*(none)*
Subclause 19.1.10, “ROUTINE_PRIVILEGES view”	*(none)*
Subclause 19.1.11, “ROUTINE_TABLE_USAGE view”	*(none)*
Subclause 19.1.12, “ROUTINE_COLUMN_USAGE view”	*(none)*
Subclause 19.1.13, “Definition of SQL built-in functions”	*(none)*
Subclause 19.2, “Definition Schema”	Subclause 21.3, "Definition Schema"
Subclause 19.2.1, “SCHEMATA base table”	Subclause 21.3.4, "SCHEMATA base table"
Subclause 19.2.2, “DATA_TYPE_DESCRIPTOR base table”	Subclause 21.3.5, "DATA_TYPE_DESCRIPTOR base table"
Subclause 19.2.3, “MODULES base table”	*(none)*
Subclause 19.2.4, “ROUTINES base table”	*(none)*
Subclause 19.2.5, “PARAMETERS base table”	*(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 ISO/IEC 9075:1992
Subclause 19.2.6, "MODULE_TABLE_USAGE base table"	*(none)*
Subclause 19.2.7, "MODULE_COLUMN_USAGE base table"	*(none)*
Subclause 19.2.8, "MODULE_PRIVILEGES base table"	*(none)*
Subclause 19.2.9, "ROUTINE_PRIVILEGES base table"	*(none)*
Subclause 19.2.10, "ROUTINE_TABLE_USAGE base table"	*(none)*
Subclause 19.2.11, "ROUTINE_COLUMN_ USAGE base table"	*(none)*
Clause 20, "Status codes"	Clause 22, "Status codes"
Subclause 20.1, "SQLSTATE"	Subclause 22.1, "SQLSTATE"
Clause 21, "Conformance"	Clause 23, "Conformance"
Subclause 21.1, "Claims of conformance"	Subclause 23.2, "Claims of conformance"
Subclause 21.2, "Extensions and options"	Subclause 23.3, "Extensions and options"
Subclause 21.2.1, "Information Schema require-ments"	*(none)*
Subclause 21.2.2, "Schema manipulation require-ments"	*(none)*
Subclause 21.3, "Flagger requirements"	Subclause 23.4, "Flagger requirements"
Annex A, "Implementation-defined elements"	Annex B, "Implementation-defined elements"
Annex B, "Implementation-dependent elements"	Annex C, "Implementation-dependent elements"
Annex C, "Deprecated features"	Annex D, "Deprecated features"
Annex D, "Incompatibilities with ISO/IEC 9075:1992"	Annex E, "Incompatibilities with ISO/IEC 9075:1989"
Table 1, "Clause, Subclause, and Table relation-ships"	*(none)*
Table 2, "Standard programming languages"	Table 16, "Standard programming languages"
Table 3, "Data type correspondences for Ada"	*(none)*
Table 4, "Data type correspondences for C"	*(none)*
Table 5, "Data type correspondences for COBOL"	*(none)*
Table 6, "Data type correspondences for Fortran"	*(none)*
Table 7, "Data type correspondences for MUMPS"	*(none)*
Table 8, "Data type correspondences for Pascal"	*(none)*
Table 9, "Data type correspondences for PL/I"	*(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 ISO/IEC 9075:1992
Table 10, "Data types of <key word>s used in SQL item descriptor areas"	Table 17, "Data types of <key word>s used in SQL item descriptor areas"
Table 11, "Codes used for input/output SQL parameter modes in Dynamic SQL"	*(none)*
Table 12, "<identifier>s for use with <get diagnostics statement>"	Table 21, "<identifier>s for use with <get diagnostics statement>"
Table 13, "SQL-statement character codes for use in the diagnostics area"	Table 22, "SQL-statement character codes for use in the diagnostics area"
Table 14, "SQLSTATE class and subclass values"	Table 23, "SQLSTATE class and subclass values"

3.4 Object identifier for Database Language SQL

Function

The object identifier for Database Language SQL identifies the characteristics of an SQL-implementation to other entities in an open systems environment.

Format

```
<SQL object identifier> ::=
        <SQL provenance> <SQL variant>
      | <SQL provenance> <SQL variant> <SQL parts>

<SQL parts> ::=
      4 | sqlpsm1996 <left paren> 4 <right paren>
```

Syntax Rules

1) Insert this SR <SQL parts> shall not be specified if the <SQL edition> is specified as <1987> or <1989>.

4 Concepts

4.1 SQL-server Modules

An *SQL-server module* is a persistent object defined in a schema and identified by an <SQL-server module name>. SQL-server modules are created with <SQL-server module definition>s and destroyed with <drop module statement>s and by <drop schema statement>s that destroy the schemas that contain them.

An <SQL-server module definition> contains an <SQL-server module name>, an optional <SQL-server module character set specification>, an optional <SQL-server module schema clause>, an optional <SQL-server module path specification>, zero or more declared local temporary tables specified by <temporary table declaration>s, and one or more <SQL-invoked routine>s. The <SQL-server module name> of an SQL-server module is a <qualified name>. The character set specified by the <SQL-server module character set specification> identifies the character repertoire used for expressing the names of schema objects used in the <SQL-server module definition>. The <default schema name> specified by the <SQL-server module schema clause> identifies the schema name used for implicit qualification of unqualified names appearing in the <SQL-server module definition>. The SQL-invoked routines of an SQL-server module are invoked only from SQL-statements.

An SQL-server module has an *SQL-server module authorization identifier*, which is set to the authorization identifier of the owner of the schema that contains the SQL-server module at the time the SQL-server module is created. The SQL-server module authorization identifier acts as the current <authorization identifier> for privilege determination for the SQL objects, if any, contained in the SQL-server module.

An SQL-server module is described by an SQL-server module descriptor. An SQL-server module descriptor includes:

— The SQL-server module name of the SQL-server module.

— The descriptor of the character set used for representing the SQL-server module's <identifier>s and <character string literal>s.

— The default schema name used for implicit qualification of unqualified names in the SQL-server module.

— The SQL-server module authorization identifier of the SQL-server module.

— The list of schema names contained in the <SQL-server module path specification>.

— The table descriptor of every local temporary table declared in the SQL-server module.

— The descriptor of every SQL-invoked routine contained in the SQL-server module.

— The text of the <SQL-server module definition>.

4.2 SQL-invoked routines

An *SQL-invoked routine* is an SQL-invoked procedure or an SQL-invoked function. An SQL-invoked routine comprises at least a <routine name>, a sequence of <SQL parameter declaration>s, and a <routine body>.

An SQL-invoked routine may be an element of an SQL-schema or of an SQL-server module. An SQL-invoked routine that is an element of an SQL-schema is called a *schema-level routine*.

An *SQL-invoked procedure* is an SQL-invoked routine that is invoked from an SQL <call statement>. An SQL-invoked procedure may have input SQL parameters, output SQL parameters, and SQL parameters that are both input SQL parameters and output SQL parameters. The format of an SQL-invoked procedure is specified by <SQL-invoked procedure> (see Subclause 10.18, "<SQL-invoked routine>").

An *SQL-invoked function* is an SQL-invoked routine that is invoked as part of an SQL <value expression>. An SQL-invoked function has only input SQL parameters and returns its results as the value of its invocation. The format of an SQL-invoked function is specified by <SQL-invoked function> (see Subclause 10.18, "<SQL-invoked routine>").

An SQL-invoked routine can be an *SQL routine* or an *external routine*. An SQL routine is an SQL-invoked routine whose <language clause> specifies SQL. The <routine body> of an SQL routine is an <SQL procedure statement>; the <SQL procedure statement> forming the <routine body> can be any SQL-statement, including an <SQL control statement>, but excluding an <SQL schema statement>, <SQL connection statement>, <SQL dynamic statement>, or <SQL transaction statement>.

An external routine is one whose <language clause> does not specify SQL. The <routine body> of an external routine is an <external body reference> whose <external routine name> identifies a program written in some standard programming language other than SQL.

An SQL-invoked routine is uniquely identified by a <specific name>, called the *specific name* of the SQL-invoked routine.

An invocation of an SQL-invoked routine specifies the <routine name> of the SQL-invoked routine and supplies a sequence of SQL argument values corresponding to the <SQL parameter declaration>s of the SQL-invoked routine. A *subject routine* is the SQL-invoked routine that may be invoked by a <routine invocation>. Such an invocation assigns the SQL arguments to the SQL parameters of one of the subject routines and causes the <routine body> of that subject routine to be executed. If the SQL-invoked routine is an SQL routine, then the <routine body> is an <SQL procedure statement> that is executed according to the General Rules of <SQL procedure statement>. If the SQL-invoked routine is an external routine, then the <routine body> identifies a program written in some standard programming language other than SQL that is executed according to the rules of that standard programming language.

The <routine body> of an SQL-invoked routine is always executed under the same SQL-session from which the SQL-invoked routine was invoked. Before the execution of the <routine body>, a new context for the current SQL-session is created and the values of the current context preserved. When the execution of the <routine body> completes the original context of the current SQL-session is restored.

If the SQL-invoked routine is an external routine, then an effective SQL parameter list is constructed before the execution of the <routine body>. The effective SQL parameter list has different entries depending on the parameter passing style of the SQL-invoked routine. The value of each entry in the effective SQL parameter list is set according to the General Rules of Subclause 9.1, "<routine invocation>", and passed to the program identified by the <routine body> according to the rules of Subclause 11.5, "Data type correspondences". After the execution of that program, if the

parameter passing style of the SQL-invoked routine is SQL, then the SQL implementation obtains the values for output parameters (if any), the value (if any) returned from the program, the value of the SQLSTATE, and the value of the message text (if any) from the values assigned by the program to the effective SQL parameter list. If the parameter passing style of the SQL-invoked routine is GENERAL, then such values are obtained in an implementation-defined manner.

Different SQL-invoked routines can have equivalent <routine name>s. No two SQL-invoked functions in the same schema are allowed to have the same signature. No two SQL-invoked procedures in the same schema are allowed to have the same name and the same number of parameters. *Subject routine determination* is the process for choosing the actual subject routine to be invoked for a given invocation of an SQL-invoked routine with a given <routine name>. Subject routine determination for SQL-invoked functions considers the data types of *all* of the arguments to the invocation of the SQL-invoked funtion in order from left to right. Where there is not an exact match between the data types of the arguments and the data types of the parameters, type precedence lists are used to determine the closest match. See Subclause 8.1, "Subject routine determination".

If a <routine invocation> is contained in a <query expression> of a view, a check constraint or an assertion, or in an <SQL-invoked routine>, then the subject routines for that invocation are determined at the time the view is created, the check constraint is defined, the assertion is created, or the SQL-invoked routine is created. Thus, the same SQL-invoked routines are invoked whenever the view is used, the check constraint or assertion is evaluated, or the SQL-invoked routine is invoked.

All <item reference>s in the <routine body> of an SQL routine are resolved to identify a particular column, SQL parameter, or SQL variable at the time that the SQL routine is created. Thus, the same columns, SQL parameters, and SQL variables are referenced whenever the SQL routine is invoked.

An SQL-invoked routine is either *deterministic* or *possibly non-deterministic*. An SQL-invoked function that is deterministic always returns the same return value for a given list of SQL argument values. An SQL-invoked procedure that is deterministic always returns the same values in its output and inout SQL parameters for a given list of SQL argument values. An SQL-invoked routine is possibly non-deterministic if, during invocation of that SQL-invoked routine, an implementation might, at two different times when the state of the SQL-data is the same, produce unequal results due to General Rules that specify implementation-dependent behavior.

An external routine either *does not possibly contain SQL* or *possibly contains SQL*. Only an external routine that possibly contains SQL may execute SQL-statements during its invocation.

An SQL-invoked routine may or may not *possibly read SQL-data*. Only an SQL-invoked routine that possibly reads SQL-data may read SQL-data during its invocation.

An SQL-invoked routine may or may not *possibly modify SQL-data*. Only an SQL-invoked routine that possibly modifies SQL-data may modify SQL-data during its invocation.

An SQL-invoked routine has a *routine authorization identifier*, which is (directly or indirectly) the authorization identifier of the owner of the schema that contains the SQL-invoked routine at the time that the SQL-invoked routine is created. An SQL-invoked routine that is an external routine also has an *external routine authorization identifier*, which is the <module authorization identifier>, if any, of the <module> contained in the external program identified by the <routine body> of the external routine. If that <module> does not specify a <module authorization identifier>, then the *external routine authorization identifier* is an implementation-defined authorization identifier.

For SQL routines, the routine authorization identifier acts as the current <authorization identifier> for privilege determination for the SQL objects, if any, contained in the SQL routine during the execution of that SQL routine. For external routines, the external routine authorization identifier

acts as the current <authorization identifier> for privilege determination for the SQL objects, if any, contained in the external routine during the execution of that external routine.

An SQL-invoked routine has a *routine SQL-path*, which is inherited from its containing SQL-server module or schema, the current SQL-session, or the containing module.

An SQL-invoked routine that is an external routine also has an *external routine SQL-path*, which is derived from the <module path specification>, if any, of the <module> contained in the external program identified by the routine body of the external routine. If that <module> does not specify a <module path specification>, then the *external routine SQL-path* is an implementation-defined SQL-path. For both SQL and external routines, the SQL-path of the current SQL-session is used to determine the search order for the subject routines of a <routine invocation> whose <routine name> does not contain a <schema name> if the <routine invocation> is contained in a <preparable statement> or in a <direct SQL statement>. SQL routines use the routine SQL-path to determine the search order for the subject routines of a <routine invocation> whose <routine name> does not contain a <schema name> if the <routine invocation> is not contained in a <preparable statement> or in a <direct SQL statement>. External routines use the external routine SQL-path to determine the search order for the subject routines of a <routine invocation> whose <routine name> does not contain a <schema name> if the <routine invocation> is not contained in a <preparable statement> or in a <direct SQL statement>.

An SQL-invoked routine is described by a routine descriptor. A routine descriptor contains:

— The routine name of the SQL-invoked routine.

— The <specific name> of the SQL-invoked routine.

— The routine authorization identifier of the SQL-invoked routine.

— The routine SQL-path of the SQL-invoked routine.

— The name of the language in which the body of the SQL-invoked routine is written.

— For each of the SQL-invoked routine's SQL parameters, the <SQL parameter name>, if it is specified, the <data type>, the ordinal position, and an indication of whether the SQL parameter is an input SQL parameter, an output SQL parameter, or both an input SQL parameter and an output SQL parameter.

— An indication of whether the SQL-invoked routine is an SQL-invoked function or an SQL-invoked procedure.

— An indication of whether the SQL-invoked routine is a schema-level routine.

— An indication of whether the SQL-invoked routine is deterministic or possibly non-deterministic.

— Indications of whether the SQL-invoked routine possibly modifies SQL-data, possibly reads SQL-data, possibly contains SQL, and does not possibly contain SQL.

— If the SQL-invoked routine is an SQL-invoked function, then the <returns data type> of the SQL-invoked function.

— If the SQL-invoked routine is an SQL routine, then the SQL routine body of the SQL-invoked routine.

— If the SQL-invoked routine is an external routine, then:

  • The <external routine name> of the external routine.

  • The <parameter style> of the external routine.

  • If the external routine specifies a <result cast>, then an indication that it specifies a <result cast> and the <data type> specified in the <result cast>.

  • The external routine authorization identifier of the external routine.

  • The external routine SQL-path of the external routine.

  • The effective SQL parameter list of the external routine.

— If the SQL-invoked routine is a schema-level routine, then the <schema name> of the schema that includes the SQL-invoked routine.

— If the SQL-invoked routine is not a schema-level routine, then the <SQL-server module name> of the SQL-server module that includes the SQL-invoked routine and the <schema name> of the schema that includes the SQL-server module.

4.3 SQL-paths

An SQL-path is a list of one or more <schema name>s that determines the search order for the subject routine of a <routine invocation> whose <routine name> does not contain a <schema name>. If the specification of an SQL-path does not specify a schema whose schema name is INFORMATION_SCHEMA, then INFORMATION_SCHEMA is assumed to precede all of the specified schema names.

The value specified by CURRENT_PATH is the value of the SQL-path of the current SQL-session. This SQL-path is used to search for subject routines of a <routine invocation> whose <routine name> does not contain a <schema name> when the <routine invocation> is contained in <preparable statement>s that are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement>, or contained in <direct SQL statement>s that are invoked directly. The definition of SQL-schemas and SQL-server modules specify an SQL-path that is used to search for subject routines of a <routine invocation> whose <routine name>s do not contain a <schema name> when the <routine invocation> is contained respectively in the <schema definition> or the <SQL-server module definition>.

4.3.1 Type conversions and mixing of data types

insert this paragraph Two data types are said to be *compatible* if they are mutually assignable and their descriptors include the same data type name.

NOTE 3 – The data types `CHARACTER(n) CHARACTER SET` *`some-character-set`* and `CHARACTER(n) CHARACTER SET` *`different-character-set`* have descriptors that include the same data type name (CHARACTER), but are not mutually assignable; therefore, they are not compatible.

4.4 Tables

Replace 16th paragraph A declared local temporary table may be declared in a module or in an SQL-server module.

Insert after 16th paragraph A declared local temporary table that is declared in a module is a named table defined by a <temporary table declaration> that is effectively materialized the first time any <procedure> in the <module> that contains the <temporary table declaration> is executed. A declared local temporary table is accessible only by <procedure>s in the <module> that contains the <temporary table declaration>. The effective <schema name> of the <qualified name> of the declared local temporary table may be thought of as the implementation-dependent SQL-session identifier associated with the SQL-session and a unique implementation-dependent name associated with the <module> that contains the <temporary table declaration>.

Insert after 16th paragraph A declared local temporary table that is declared in an SQL-server module is a named table defined by a <temporary table declaration> that is effectively materialized the first time any <module routine> in the <SQL-server module definition> that contains the <temporary table declaration> is executed.

Insert after 16th paragraph A declared local temporary table is accessible only by <module routine>s in the <SQL-server module definition> that contains the <temporary table declaration>. The effective <schema name> of the <qualified name> of the declared local temporary table may be thought of as the implementation-dependent SQL-session identifier associated with the SQL-session and the name of the <SQL-server module definition> that contains the <temporary table declaration>.

Insert after 16th paragraph All references to a declared local temporary table are prefixed by "MODULE.".

4.5 Integrity constraints

Replace 2nd paragraph A <query expression> or <query specification> is *possibly non-deterministic* if, when evaluating that <query expression> or <query specification>, an implementation might, at two different times when the state of the SQL-data is the same, produce unequal results due to General Rules that specify implementation-dependent behavior.

Replace 3rd paragraph No integrity constraint shall be defined using a possibly non-deterministic <query specification> or <query expression>, or a <routine invocation> whose subject routines include an SQL-invoked routine that is a possibly non-deterministic routine.

4.6 SQL-schemas

Replace 2nd paragraph In this International standard, the term "schema" is used only in the sense of SQL-schema. Each component descriptor is either a domain descriptor, a base table descriptor, a view descriptor, a constraint descriptor, a privilege descriptor, a character set descriptor, a collation descriptor, a translation descriptor, an SQL-server module descriptor or an SQL-invoked routine descriptor. The persistent objects described by the descriptors are said to be *owned by* or to have been *created by* the <authorization identifier> of the schema.

An SQL-schema additionally includes:

— The SQL-path of the SQL-schema.

4.7 Parameters

4.7.1 Status parameters

Insert this paragraph Exception conditions or completion conditions may be raised during the execution of an <SQL procedure statement>. One of the conditions becomes the active condition when the <SQL procedure statement> terminates; the *active condition* is the condition returned in SQLSTATE. If the active condition is an exception condition, then it is called the *active exception condition*. If the active condition is a completion condition, then it is called the *active completion condition*.

Insert this paragraph If the <SQL procedure statement> is a <compound statement>, then the active condition may result from the action of some exception handler specified in the <compound statement>.

4.8 Diagnostics area

Insert this paragraph An implementation shall place information about a completion or exception condition that causes a handler to be activated into the diagnostics area prior to activating the handler. If other conditions are raised, then it is implementation-defined whether the implementation places information about them into the diagnostics area.

Insert this paragraph The diagnostics area is emptied during the execution of a <signal statement>. Information is added to the diagnostics area during the execution of a <resignal statement>.

4.9 Cursors

Insert this paragraph For every <declare cursor> in a <compound statement>, a cursor is effectively created each time the <compound statement> is executed, and destroyed when that execution completes.

4.10 Condition handling

Condition handling is the method of handling exception and completion conditions in SQL/PSM. Condition handling provides a *<handler declaration>* to define a handler, specifying its type, the exception and completion conditions it can resolve, and the action it takes to do so. Condition handling also provides the ability to explicitly signal exception and completion conditions.

<handler declaration>s specify the handling of exception and completion conditions. <handler declaration>s can be specified in <compound statement>s. The scope of a <handler declaration> specified in a <compound statement> is that <compound statement> excluding every <SQL schema statement> contained in that <compound statement>.

A <handler declaration> associates one or more conditions with a handler action. The handler action is an <SQL procedure statement>.

A *general <handler declaration>* is one that is associated with the SQLEXCEPTION, SQLWARNING, or NOT FOUND SQLSTATE class. All other <handler declaration>s are *specific <handler declaration>*s.

4.10 Condition handling

A condition represents an error or informational state caused by execution of an <SQL procedure statement>. Conditions are raised to provide information in the diagnostics area about the execution of an <SQL procedure statement>.

A <condition declaration> is used to declare a <condition name>, and to optionally associate it with an SQLSTATE value. If a <condition declaration> does not specify an SQLSTATE value, it declares a *user-defined exception condition*. <condition name>s can be used in <handler declaration>s, <signal statement>s, and <resignal statement>s.

When the <compound statement> containing a <handler declaration> is executed, a handler is created for the conditions associated with that <handler declaration>. A created handler is *activated* when it is the most appropriate handler for an exception or completion condition that has been raised by an SQL-statement. Such a handler is an *active* handler.

The most appropriate handler is determined during execution of an implicit or explicit <resignal statement>. An implicit <resignal statement> is executed when a <compound statement> or <handler action> completes with a condition other than *successful completion*.

If there is no most appropriate handler and the condition is an exception condition, then the SQL-statement raising the exception condition is terminated with that exception condition. This type of exception condition is called an *unhandled exception condition*. Unhandled exception conditions are examined at the next visible scope for handling. If an exception condition remains unhandled at the outer <procedure> or <direct SQL statement>, it is seen by the SQL-client. Even if the SQL-client resolves the exception condition, execution is not resumed in the SQL-server where the exception condition was raised.

If there is no most appropriate handler and the condition is a completion condition, then execution is resumed as specified in Subclause 3.3.4.1, "Exceptions", in ISO/IEC 9075:1992. This type of completion condition is called an *unhandled completion condition*.

A handler type specifies CONTINUE, EXIT, or UNDO.

If a handler type specifies CONTINUE, then, when the handler is activated, it will:

— Execute the handler action.

— Return control to the <compound statement> from which it was invoked and execute all other SQL-statements following the one that raised the condition.

If a handler type specifies EXIT, then, when the handler is activated, it will:

— Execute the handler action.

— Implicitly LEAVE the <compound statement> for which the handler was created, with no active exception condition.

If a handler type specifies UNDO, then, when the handler is activated, it will:

— Roll back all of the changes to SQL-data or to schemas by the execution of every SQL-statement contained in the SQL-statement list of the <compound statement> at the scope of the handler and cancel any <SQL procedure statement>s triggered by the execution of such statements.

— Execute the handler action.

— Return control to the end of the <compound statement> for which the handler was created.

If a <handler action> completes with a completion condition: *successful completion*, then it was able to resolve the condition, and execution resumes as specified in Subclause 13.4, "<handler declaration>".

If a <handler action> completes with an exception or completion condition other than *successful completion*, then an implicit <resignal statement> is executed. The <resignal statement> determines whether there is another <handler declaration> that can resolve the condition.

4.11 SQL-statements

4.11.1 SQL-statements classified by function

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

— SQL-control statements

— SQL-control declarations

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

— <SQL-server module definition>

— <drop module statement>

— <schema routine>

— <drop routine statement>

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

— <call statement>

— <return statement>

— <compound statement>

— <case statement>

— <if statement>

— <leave statement>

— <loop statement>

— <while statement>

— <repeat statement>

— <for statement>

— <assignment statement>

Insert this paragraph The following are the SQL-control declarations:

— <condition declaration>

— <handler declaration>

— <SQL variable declaration>

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

— <set path statement>

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

— <signal statement>

— <resignal statement>

4.11.2 Embeddable SQL-statements

Insert this paragraph The following are additional SQL-statements that are embeddable in an <embedded SQL host program> and that may be the <SQL procedure statement> in <procedure> in a <module>:

— All SQL-control statements

4.11.3 Preparable and immediately executable SQL-statements

Insert this paragraph The following are additional SQL-statements that are preparable and immediately executable:

— <call statement>

4.11.4 Directly executable SQL-statements

Insert this paragraph The following are additional SQL-statements that may be executed directly:

— All SQL-control statements

4.11.5 SQL-statements and transaction states

Insert this paragraph The following additional SQL-statements are transaction-initiating SQL-statements:

- <for statement>

Insert this paragraph The following additional SQL-statement is not a transaction-initiating SQL-statement:

- <leave statement>

Insert this paragraph The following additional SQL-statements are possibly transaction-initiating SQL-statements:

- SQL-control statements other than:
  - <for statement>

- <leave statement>

Insert this paragraph If an <SQL control statement> causes the evaluation of a <subquery> and there is no current SQL-transaction, then an SQL-transaction is initiated before evaluation of the <subquery>.

Insert this paragraph If the initiation of an SQL-transaction occurs in an atomic execution context, and an SQL-transaction has already been completed in this atomic execution context, then an exception condition is raised: *invalid transaction initiation*.

4.11.6 Compound statements

A compound statement allows a sequence of SQL-statements to be considered as a single SQL-statement. A compound statement also defines a local scope in which SQL-variables, condition handlers, and cursors can be declared. See Subclause 13.3, "<compound statement>".

4.11.7 SQL-statement atomicity

The execution of all SQL-statements other than SQL-control statements is atomic. The execution of <compound statement>s that specify ATOMIC is atomic. Such an SQL-statement is called an *atomic SQL-statement*.

An atomic execution context is said to be *active* during the execution of an atomic SQL-statement or evaluation of a <subquery>. Within one atomic execution context, another atomic execution context may become active.

An SQL-transaction cannot be explicitly terminated within an atomic execution context. If the execution of an atomic SQL-statement is unsuccessful, then the changes to SQL-data or schemas made by the SQL-statement are canceled.

4.12 Privileges

Insert this paragraph A privilege further authorizes a given category of <action> to be performed on a specified SQL-invoked routine or SQL-server module by a specified <authorization identifier>.

Insert this paragraph An additional <action> that can be specified is:

— EXECUTE

Insert this paragraph A privilege descriptor with an action of EXECUTE is called an *execute privilege descriptor* and identifies the existence of a privilege on the SQL-server module or SQL-invoked routine identified by the privilege descriptor.

Insert this paragraph The identification included in an EXECUTE privilege descriptor may also identify the SQL-server module or SQL-invoked routine described by the descriptor.

Insert this paragraph Individual SQL-invoked routines contained in an SQL-server module cannot be associated with EXECUTE privilege descriptors. Only schema-level routines and SQL-server modules are associated with EXECUTE privilege descriptors.

NOTE 4 – "schema-level routine" is defined in Subclause 10.18, "<SQL-invoked routine>".

4.13 SQL-sessions

Insert this paragraph An SQL-session has an 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 At any time during an SQL-session, *containing SQL* is said to be *permitted* or *not permitted*. Similarly, *reading SQL-data* is said to be *permitted* or *not permitted* and *modifying SQL-data* is either *permitted* or *not permitted*.

Insert this paragraph The SQL-session context additionally contains:

— The text defining the current default SQL-path.

— A *routine execution context*, comprising:

- An indication as to whether or not an SQL-invoked routine is active.
- An indication as to whether or not containing SQL is permitted.
- An indication as to whether or not reading SQL-data is permitted.
- An indication as to whether or not modifying SQL-data is permitted.
- An identification of the SQL-invoked routine that is active.
- The routine SQL-path derived from the routine SQL-path if the SQL-invoked routine that is active is an SQL routine and from the external routine SQL-path if the SQL-invoked routine that is active is an external routine.

Insert this paragraph An SQL-invoked routine is *active* as soon as an SQL-statement executed by an SQL-agent causes invocation of an SQL-invoked routine and ceases to be active when execution of that invocation is complete.

Insert this paragraph When an SQL-agent causes the invocation of an SQL-invoked routine, a new context for the current SQL-session is created and the values of the current context are preserved. When the execution of that SQL-invoked routine completes, the original context of the current SQL-session is restored and some SQL-session attributes are reset.

Insert this paragraph When the routine execution context of the SQL-session indicates that an SQL-invoked routine is active, then the routine SQL-path included in the routine execution context of the SQL-session is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are not contained in a <preparable statement> or in a <direct SQL statement>.

5 Lexical elements

5.1 <token> and <separator>

Function

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

Format

```
<non-reserved word> ::=
         !! All alternatives from ISO/IEC 9075:1992

       | ATOMIC

       | CONDITION_IDENTIFIER

       | GENERAL

       | MODIFIES

       | PARAMETER_NAME

       | READS | ROUTINE_CATALOG | ROUTINE_NAME | ROUTINE_SCHEMA

       | SPECIFIC_NAME | STYLE

<reserved word> ::=
         !! All alternatives from ISO/IEC 9075:1992

       | CALL | CONDITION | CONTAINS | CURRENT_PATH

       | DETERMINISTIC | DO

       | ELSEIF | EXIT

       | FUNCTION

       | HANDLER

       | IF | INOUT

       | LEAVE | LOOP

       | OUT

       | PARAMETER | PATH

       | REPEAT | RESIGNAL | RETURN | RETURNS | ROUTINE

       | SIGNAL | SPECIFIC | SQLEXCEPTION | SQLWARNING
```

```
| UNDO | UNTIL

| WHILE
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

5.2 Names and identifiers

Function

Specify names.

Format

```
<SQL-server module name> ::=
      <qualified name>

<SQL variable name> ::=
      <identifier>

<condition name> ::=
      <identifier>

<routine name> ::=
      <qualified name>

<specific name> ::=
      <qualified name>

<external routine name> ::=
        <identifier>
      | <character string literal>

<SQL parameter name> ::=
      <identifier>
```

Syntax Rules

1) Replace SR1) If a <character set specification> is not specified in an <identifier>, then the set of characters contained in the <identifier> shall be wholly contained in either <SQL language character> or the character repertoire indicated by:

   Case:

   a) If the <identifier> is contained, without an intervening <schema definition> or <SQL-server module 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 character set name for the SQL-session,

   b) If the <identifier> is contained in an <SQL-server module definition>, then the <SQL-server module character set specification>,

   c) If the <identifier> is contained in a <schema definition>, then the <schema character set specification>,

   d) If the <identifier> is contained in a <module>, then the <module character set specification>.

2) Replace SR8) If a <qualified name> does not contain a <schema name>, then

Case:

a) If the <qualified name> is contained, without an intervening <schema definition> or <SQL-server module 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.

b) If the <qualified name> is contained in an <SQL-server module definition>, then the <default schema name> that is specified or implicit in the <SQL-server module definition> is implicit.

c) If the <qualified name> is contained in a <schema definition>, then the <schema name> that is specified or implicit in the <schema definition> is implicit.

d) If the <qualified name> is contained in a <module>, then the <schema name> that is specified or implicit for the <module> is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR10) A <parameter name> identifies a host parameter.

2) Insert after GR10) An <SQL parameter name> identifies an SQL parameter.

3) Insert this GR An <SQL-server module name> identifies an SQL-server module.

4) Insert this GR An <SQL variable name> identifies an SQL variable.

5) Insert this GR A <condition name> identifies an exception condition or a completion condition and optionally a corresponding SQLSTATE value.

6) Insert this GR A <routine name> identifies one or more SQL-invoked routines.

7) Insert this GR A <specific name> identifies an SQL-invoked routine.

8) Insert this GR An <external routine name> identifies a program written in a standard programming language other than SQL.

6 Scalar expressions

6.1 <value specification> and <target specification>

Function

Specify one or more values, parameters, or variables.

Format

```
<general value specification> ::=
         !! All alternatives from ISO/IEC 9075:1992
       | CURRENT_PATH
       | <routine invocation>
       | <SQL parameter or SQL variable reference>

<simple value specification> ::=
         !! All alternatives from ISO/IEC 9075:1992
       | <SQL parameter or SQL variable reference>

<target specification> ::=
         !! All alternatives from ISO/IEC 9075:1992
       | <SQL parameter or SQL variable reference>

<simple target specification> ::=
         !! All alternatives from ISO/IEC 9075:1992
       | <SQL parameter or SQL variable reference>
```

Syntax Rules

1) Insert this SR If a <routine invocation> is specified, then the subject SQL-invoked routine shall be an SQL-invoked function.

2) Insert this SR The data type of CURRENT_PATH is character string. Whether the character string is fixed-length or variable-length, and its length if it is fixed-length or its maximum length if it is variable-length, are implementation-defined. The character set of the character string is SQL_TEXT.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR The value specified by CURRENT_PATH is the value of the SQL-path of the current SQL-session.

6.1 <value specification> and <target specification>

2) Insert this GR The value specified by CURRENT_PATH is a <schema name list> where <catalog name>s are <delimited identifier>s and the <unqualified schema name>s are <delimited identifier>s. Each <schema name> is separated from the preceding <schema name> by a <comma> with no intervening <space>s.

3) Replace GR12) If a <simple value specification> evaluates to the null value, then an exception condition is raised: *data exception — null value not allowed*.

6.2 <column reference>

Function

Reference a column

Format

```
<column reference> ::=
      <item reference>
```

Syntax Rules

1) Replace SR1) A <column reference> *CR* shall be an <item reference> that is a column reference. Let *CN* be the <column name> contained in *CR*.

2) Replace SR2) If *CR* contains a <qualifier>, let *Q* be that <qualifier>.

3) Replace SR3) through SR7) If *CR* does not contain a <qualifier>, then let *Q* be the implicit qualifier determined by the Syntax Rules of Subclause 6.3, "<item reference>", in this part of ISO/IEC 9075.

Access Rules

No additional Access Rules.

General Rules

1) Delete all GRs

6.3 <item reference>

Function

Reference a column, an SQL parameter, or an SQL variable.

Format

```
<item reference> ::=
       [ <item qualifier> <period> ] <item name>

<item qualifier> ::=
       <qualifier>

<item name> ::=
         <column name>
       | <SQL parameter name>
       | <SQL variable name>

<SQL parameter or SQL variable reference> ::=
       <item reference>
```

Syntax Rules

1) Let *IR* be an <item reference> and let *IN* be the <item name> contained in *IR*.

2) If *IR* contains an <item qualifier> *IQ*, then *IR* shall appear within the scope of one or more exposed <table name>s or <correlation name>s that are equivalent to *IQ*. If there is more than one such exposed <table name> or <correlation name>, then the one with the innermost scope is specified.

   a) Let *V* be the table identified by *IQ*.

   b) *V* shall include a column whose <column name> is equivalent to *IN*.

   c) If *V* is the table identified by a <table reference> in a <joined table> *JT*, then *IN* shall not be a common column name in *JT*.

      NOTE 5 – "Common column name" is defined in Subclause 7.5, "<joined table>", in ISO/IEC 9075:1992.

3) If *IR* does not contain an <item qualifier>, then *IR* shall be contained within the scope of one or more exposed <table name>s or <correlation name>s whose associated tables include a column whose <identifier> is equivalent to *IN* or within the scope of one or more <routine name>s or <beginning label>s whose associated <SQL parameter list> or <local declaration list> includes an SQL parameter or SQL variable whose <identifier> is equivalent to *IN*. Let the phrase *possible qualifiers* denote those exposed <table name>s, <correlation name>s, <routine name>s, and <statement label>s.

   a) Case:

      i) If the innermost scope contains exactly one possible qualifier, then the qualifier *IQ* equivalent to that unique exposed <table name>, <correlation name>, <routine name>, or <beginning label> is implicit.

ii) If there is more than one possible qualifier within the innermost scope, then:

1) Each possible qualifier shall be a <table name> or a <correlation name> of a <table reference> that is directly contained in a <joined table> *JT*.

2) *IN* shall be a common column name in *JT*.

NOTE 6 – "Common column name" is defined in Subclause 7.5, "<joined table>", in ISO/IEC 9075:1992.

3) The implicit qualifier *IQ* is implementation-dependent. The scope of *IQ* is that which *IQ* would have had if *JT* had been replaced by the <table reference>:

```
(JT) AS IQ
```

b) Let *V* be the table, SQL parameter list, or <local declaration list> associated with *IQ*.

4) Case:

a) If *IQ* is a <table name> or <correlation name>, then *IR* is a *column reference*. *IN* shall uniquely identify a column of the table *V*. Let *R* be that column.

b) If *IQ* is the <routine name> of an SQL-invoked routine, then

Case:

i) If *IN* is an SQL variable of *V*, then *IR* is an *SQL variable reference*. Let *R* be that SQL variable.

ii) Otherwise, *IR* is an *SQL parameter reference*. *IN* is an SQL parameter of *V*. Let *R* be that SQL parameter.

c) If *IQ* is a <beginning label>, then *IR* is an *SQL variable reference*. *IN* is an SQL variable of *V*. Let *R* be that SQL variable.

5) The data type of *IR* is the data type of *R*.

6) If the data type of *IR* is character string, then *IR* has the *Implicit* coercibility attribute and its collating sequence is the default collating sequence for the column, SQL parameter, or SQL variable *R*.

7) An <SQL parameter or SQL variable reference> shall be an <item reference> that is an SQL parameter reference or an SQL variable reference.

Access Rules

None.

General Rules

1) Depending on whether *IR* is a column reference, SQL parameter reference, or SQL variable reference, *IR* references column *IN* in a given row of *V*, SQL parameter *IN* of a given invocation of the SQL-invoked routine that contains *V*, or SQL variable *IN* of a given execution of the <compound statement> whose <local declaration list> is *V*.

6.4 <datetime value function>

Function

Specify a function yielding a value of type datetime.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR3) If an <SQL procedure statement> *S* contains, without an intervening <SQL procedure statement>, one or more <value expression>s that generally contain, without an intervening <routine invocation> whose subject routines do not include an SQL function, one or more <datetime value function>s, then all such <datetime value function>s are effectively evaluated simultaneously. The time of evaluation of a <datetime value function> during the execution of *S* is implementation-dependent.

6.5 <case expression>

Function

Specify a conditional value.

Format

No additional Format items.

Syntax Rules

1) Insert this SR If <case specification> specifies <simple case>, then the <case operand> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic or that possibly modifies SQL-data.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

6.6 <value expression>

Function

Specify a value.

Format

```
<value expression primary> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <SQL parameter or SQL variable reference>
```

Syntax Rules

1) Replace SR2) If the data type of a <value expression primary> is character string, then the collating sequence and coercibility attribute of the <value expression primary> are the collating sequence and coercibility attribute of the <unsigned value specification>, <column reference>, <SQL parameter or SQL variable reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, or <cast specification> immediately contained in the <value expression primary>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

7 Query expressions

7.1 <table value constructor>

Function

Specify a set of <row value constructor>s to be constructed into a table.

Format

No additional Format items.

Syntax Rules

1) Insert this SR A <table value constructor> is *possibly non-deterministic* if it contains a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

7.2 <table expression>

Function

Specify a table or a grouped table.

Format

No additional Format items.

Syntax Rules

1) Insert this SR A <table expression> is *possibly non-deterministic* if it contains a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

7.3 <query specification>

Function

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

Format

No additional format items.

Syntax Rules

1) Replace SR10) A column of the table that is the result of a <query specification> is *possibly nullable* if and only if it contains a <column reference> referencing some column *C* that is possibly nullable, an <indicator parameter>, an <indicator variable>, an SQL parameter, an SQL variable, a <routine invocation>, a <subquery>, `CAST NULL AS X` (where *X* represents some <data type> or <domain name>), SYSTEM_USER, A <dynamic parameter specification>, or a <set function specification> that does not contain COUNT.

2) Insert after SR13)a) The <query specification> contains a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

7.4 <query expression>

Function

Specify a table.

Format

No additional Format items.

Syntax Rules

1) Insert this SR A <query expression> *QE* shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data, unless *QE* is a <table value constructor> and is immediately contained in an <insert columns and source> that is immediately contained in an <insert statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

7.5 <scalar subquery>, <row subquery>, and <table subquery>

Function

Specify a scalar value, a row,or a table derived from a <query expression>.

Format

No additional Format items

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR1) During the evaluation of a <subquery>, an atomic execution context is active.

8 Data assignment rules and routine determination

8.1 Subject routine determination

Function

Determine the set of subject routines applicable to a given routine invocation.

Syntax Rules

1) Let *SR* and *AL* be respectively a set of SQL-invoked routines, arbitrarily ordered, and the <SQL argument list> specified in an application of this Subclause.

2) Let n be the number of SQL-invoked routines in *SR*. Let R_i be the i-th SQL-invoked routine in *SR* in the ordering of *SR*.

3) Let m be the number of SQL arguments in *AL*. Let A_j be the j-th SQL argument in *AL*.

4) Let $SDTA_j$ be the data type of A_j.

5) Let $SDTP_{i,j}$ be the data type of the j-th SQL parameter of R_i.

6) For r varying from 1 to m, if there is more than one SQL-invoked routine in *SR*, then for each pair of SQL-invoked routines { R_p, R_q } in *SR*, if $SDTP_{p,r} \prec SDTP_{q,r}$ in the type precedence list of $SDTA_r$, then eliminate R_q from *SR*.

   NOTE 7 – The "type precedence list" of a given type is determined by Subclause 8.2, "Type precedence list determination".

7) The set of subject routines is the set of SQL-invoked routines remaining in *SR*.

8.2 Type precedence list determination

Function

Determine the type precedence list of a given type.

Syntax Rules

1) Let *DT* be the data type specified in an application of this Subclause.

2) Let *TPL* be the *type precedence list* of *DT*.

3) Let "$A \prec B$" represent "*A* has precedence over *B*" and let "$A \simeq B$" represent "*A* has the same precedence as *B*".

   a) If $A \prec B$ and ($B \prec C$ or $B \simeq C$) then $A \prec C$

   b) If $A \simeq B$ and $B \simeq C$) then $A \simeq C$

   c) If $A \simeq B$ and $B \prec C$) then $A \prec C$

4) If *DT* is fixed-length character string, then *TPL* is

   `CHARACTER, CHARACTER VARYING`

5) If *DT* is variable-length character string, then *TPL* is

   `CHARACTER VARYING`

6) If *DT* is fixed-length bit string, then *TPL* is

   `BIT, BIT VARYING`

7) If *DT* is variable-length bit character string, then *TPL* is

   `BIT VARYING`

8) If *DT* is numeric, then:

   a) Let *NDT* be the following set of numeric types: SMALLINT, INTEGER, NUMERIC, DECIMAL, REAL, FLOAT, and DOUBLE PRECISION.

      If the radix of any of the exact numeric types SMALLINT, INTEGER, NUMERIC, or DECIMAL is decimal, then let "precision" mean the product of $\log_2(10)$ and the actual specified or implementation-defined precision.

   b) The following set *P* of precedence relationships holds:

      i) If the implementation-defined precision of INTEGER is greater than the implementation-defined precision of SMALLINT, then SMALLINT $\prec$ INTEGER; otherwise, SMALLINT $\simeq$ INTEGER.

      ii) Case:

         1) If the implementation-defined maximum precision of DECIMAL is greater than the implementation-defined precision of INTEGER, then INTEGER $\prec$ DECIMAL.

         2) If the implementation-defined maximum precision of DECIMAL is equal to the implementation-defined precision of INTEGER, then INTEGER $\simeq$ DECIMAL.

3) Otherwise, DECIMAL ≺ INTEGER and

Case:

A) If the implementation-defined maximum precision of DECIMAL is greater than the implementation-defined precision of SMALLINT, then SMALLINT ≺ DECIMAL,

B) If the implementation-defined maximum precision of SMALLINT is greater than the implementation-defined precision of DECIMAL, then DECIMAL ≺ SMALLINT.

C) Otherwise, SMALLINT ≃ DECIMAL.

iii) Case:

1) If the implementation-defined maximum precision of NUMERIC is greater than the implementation-defined precision of INTEGER, then INTEGER ≺ NUMERIC.

2) If the implementation-defined maximum precision of NUMERIC is equal to the implementation-defined precision of INTEGER, then INTEGER ≃ NUMERIC.

3) Otherwise, NUMERIC ≺ INTEGER and

Case:

A) If the implementation-defined maximum precision of NUMERIC is greater than the implementation-defined precision of SMALLINT, then SMALLINT ≺ NUMERIC.

B) If the implementation-defined precision of SMALLINT is greater than the implementation-defined maximum precision of NUMERIC, then NUMERIC ≺ SMALLINT.

C) Otherwise, SMALLINT ≃ NUMERIC.

iv) Case:

1) If the implementation-defined maximum precision of NUMERIC is greater than the implementation-defined maximum precision of DECIMAL, then DECIMAL ≺ NUMERIC.

2) If the implementation-defined maximum precision of NUMERIC is equal to the implementation-defined maximum precision of DECIMAL, then DECIMAL ≃ NUMERIC.

3) Otherwise, NUMERIC ≺ DECIMAL.

v) REAL ≺ DOUBLE PRECISION

vi) Case:

1) If the implementation-defined maximum precision of FLOAT is greater than the implementation-defined precision of REAL, then REAL ≺ FLOAT.

2) If the implementation-defined maximum precision of FLOAT is equal to the implementation-defined precision of REAL, then FLOAT ≃ REAL.

3) Otherwise, FLOAT $\prec$ REAL.

vii) Case:

1) If the implementation-defined maximum precision of FLOAT is greater than the implementation-defined precision of DOUBLE PRECISION, then DOUBLE PRECISION $\prec$ FLOAT.

2) If the implementation-defined maximum precision of FLOAT is equal to the implementation-defined precision of DOUBLE PRECISION, then FLOAT $\simeq$ DOUBLE PRECISION.

3) Otherwise, FLOAT $\prec$ DOUBLE PRECISION.

viii) Let *MAXEX* be whichever of INTEGER, NUMERIC, and DECIMAL has the greatest precedence and let *MINAP* be whichever of REAL and FLOAT has the least precedence. *MAXEX* $\prec$ *MINAP*.

c) Let *PTC* be {(*a*,*b*): $a \in NDT$, $b \in NDT$, $a \prec b$ or $a \simeq b$}

d) *TPL* is determined as follows:

i) *TPL* is initially empty.

ii) Let *ST* be the set of types containing *DT* and every type *T* in *NDT* for which the precedence relationship $DT \prec T$ or $DT \simeq T$ is in *PTC*.

iii) Let *n* be the number of types in *ST*.

iv) Repeat the following *n* times:

1) Let *NT* be the set of types T_k in *ST* such that there is no other type T_j in *ST* for which $T_j \prec T_k$ according to *PTC*.

2) Case:

A) If there is exactly one type T_k in *NT*, then T_k is placed next in *TPL* and all relationships of the form $T_k \prec T_r$ are removed from *PTC*, where T_r is any type in *ST*.

B) If there is more than one type T_k in *NT*, then every type T_s in *NT* is assigned the same position in *TPL* as T_k and all relationships of the forms $T_k \prec T_r$, $T_k \simeq T_r$, $T_s \prec T_r$, and $T_s \simeq T_r$ are removed from *PTC*, where T_r is any type in *ST*.

9) If *DT* is INTERVAL, then *TPL* is

DT

or a comparable interval type.

10) If *DT* is a datetime type, then *TPL* is

DT

9 Additional common elements

9.1 <routine invocation>

Function

Invoke an SQL-invoked routine.

Format

```
<routine invocation> ::=
      <routine name> <SQL argument list>

<SQL argument list> ::=
        <left paren> [ <SQL argument> [ { <comma> <SQL argument> }... ] ] <right paren>


<SQL argument> ::=
         <value expression>
       | <target specification>
```

Syntax Rules

1) Let *RN* be the <routine name> immediately contained in the <routine invocation> *RI*.

2) An SQL-invoked routine *R* is a *possibly candidate routine* for *RI* (henceforth, simply "possibly candidate routine") if and only if

   Case:

   a) If *RI* is immediately contained in a <call statement>, then *R* is an SQL-invoked procedure, and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

   b) Otherwise, *R* is an SQL-invoked function and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

3) An SQL-invoked routine *R* is an *executable routine* if and only if *R* is a possibly candidate routine and

   Case:

   a) If *R* is included in an SQL-server module *M*, then the applicable privileges include EXECUTE on *M*.

   b) Otherwise, the applicable privileges include EXECUTE on *R*.

4) Case:

   a) If <SQL argument list> does not immediately contain at least one <SQL argument>, then a *possibly invocable routine* is an executable routine that has no SQL parameters.

b) Otherwise:

i) Let *NA* be the number of <SQL argument>s in the <SQL argument list> *AL* of *RI*. Let A_i be the *i*-th <SQL argument> in *AL*.

ii) Let the *static SQL argument list* of *RI* be *AL*.

iii) Let P_i be the *i*-th SQL parameter of an executable routine. A *possibly invocable routine* is an SQL-invoked routine *R1* that is an executable routine such that:

1) *R1* has *NA* SQL parameters.

2) If *R1* is not immediately contained in a <call statement>, then for each P_i the data type of P_i shall be in the type precedence list of the data type of A_i.

NOTE 8 – "type precedence list" is defined in Subclause 8.2, "Type precedence list determination".

5) If <SQL argument list> does not immediately contain at least one <SQL argument>, then

a) Let *AL* be an empty list of SQL arguments.

b) The subject routine of *RI* is defined as follows:

i) If *RN* does not contain a <schema name>, then:

1) Case:

A) If the routine execution context of the current SQL-session indicates that an SQL-invoked routine is active, then let *DP* be the routine SQL-path of that routine execution context.

B) Otherwise,

Case:

I) If *RI* is contained in an <SQL-server module definition>, then let *DP* be the SQL-path of that <SQL-server module definition>.

II) If *RI* is contained in a <schema definition> without an intervening <SQL-server module definition>, then let *DP* be the SQL-path of that <schema definition>.

III) If *RI* 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 let *DP* be the SQL-path of the current SQL-session.

IV) Otherwise, let *DP* be the SQL-path of the <module> that contains *RI*.

2) The *subject routine* of *RI* is an SQL-invoked routine *R1* such that:

A) *R1* is a possibly invocable routine.

B) The <schema name> of the schema of *R1* is in *DP*.

C) There is no other possibly invocable routine *R2* for which the <schema name> of the schema that includes *R2* precedes in *DP* the <schema name> of the schema that includes *R1*.

ii) If *RN* contains a <schema name> *SN*, then

Case:

1) If *SN* is "INFORMATION_SCHEMA", then the single candidate routine of *RI* is the built-in function identified by <routine name>.

2) Otherwise, *SN* shall be the <schema name> of a schema *S*. The subject routine of *RI* is the possibly invocable routine (if any) contained in *S*.

c) There shall be exactly one subject routine of *RI*.

d) If *RI* is not immediately contained in a <call statement>, then the *effective returns data type* of *RI* is the data type returned by the subject routine of *RI*.

e) Let the *static SQL argument list* of *RI* be an empty list of SQL arguments.

6) If <SQL argument list> immediately contains at least one <SQL argument>, then:

a) The set of *invocable routines* of *RI* is defined as follows:

Case:

i) If *RI* is immediately contained in a <call statement>, then the *invocable routines* of *RI* are the possibly invocable routines of *RI*.

ii) Otherwise, the *invocable routines* of *RI* are those SQL-invoked routines that are possibly invocable routines such that for each P_i whose data type is character string, the character set of P_i and the character set of A_i are the same character set.

b) The set of *candidate routines* of *RI* is defined as follows:

Case:

i) If *RN* does not contain a <schema name>, then

1) Case:

A) If the routine execution context of the current SQL-session indicates that an SQL-invoked routine is active, then let *DP* be the routine SQL-path of that routine execution context.

B) Otherwise,

Case:

I) If *RI* is contained in an <SQL-server module definition>, then let *DP* be the SQL-path of that <SQL-server module definition>.

II) If *RI* is contained in a <schema definition> without an intervening <SQL-server module definition>, then let *DP* be the SQL-path of that <schema definition>.

III) If *RI* 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 let *DP* be the SQL-path of the current SQL-session.

IV) Otherwise, let *DP* be the SQL-path of the <module> that contains *RI*.

2) The candidate routines of *RI* are the set union of invocable routines of all schemas whose <schema name> is in *DP*.

ii) If *RN* contains a <schema name> *SN*, then

Case:

1) If *SN* is "INFORMATION_SCHEMA", then the single candidate routine of *RI* is the built-in function identified by <routine name>.

2) Otherwise, *SN* shall be the <schema name> of a schema *S*. The candidate routines of *RI* are the invocable routines (if any) contained in *S*.

c) The set of *compatible candidate routines* of *RI* is defined as follows.

Case:

i) If *RI* is immediately contained in a <call statement>, then the *compatible candidate routines* of *RI* are the candidate routines of *RI*.

ii) Otherwise:

1) The Syntax Rules of Subclause 8.1, "Subject routine determination", are applied to the candidate routines of *RI* and *AL*, yielding a set of subject routines *SR*.

2) There shall be at least one subject routine in *SR*.

3) Case:

A) If *RN* contains a <schema name>, then there shall be exactly one subject routine in *SR*. Let *R1* be the subject routine in *SR*.

B) Otherwise,

Case:

I) If there is exactly one subject routine in *SR*, then let *R1* be that subject routine.

II) If there is more than one subject routine in *SR*, then let *R1* be the SQL-invoked routine in *SR* such that there is no other SQL-invoked routine *R2* in *SR* for which the <schema name> of the schema that includes *R2* precedes in *DP* the <schema name> of the schema that includes *R1*.

4) Let *RT* be the data type returned by *R1*. The compatible candidate routines of *RI* are those SQL-invoked routines in the set of candidate routines of *RI* whose returned data type contains *RT* in their type precedence lists.

d) Case:

i) If *RI* is immediately contained in a <call statement>, then:

1) Let *XAL* be *AL*.

2) The subject routine *SR* of *XAL* is an SQL-invoked routine *R1* that is a compatible candidate routine of *RI* such that there is no other compatible candidate routine *R2* for which the <schema name> of the schema that includes *R2* precedes in *DP* the <schema name> of the schema that includes *R1*.

3) Let *PL* be the list of SQL parameters P_i of *SR*.

4) For each P_i that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, A_i shall be a <target specification> and P_i shall be assignable to A_i, according to the Syntax Rules of Subclause 9.1, "Retrieval assignment", in ISO/IEC 9075:1992, with A_i and P_i as *TARGET* and *VALUE*, respectively.

5) For each P_i that is an input SQL parameter or both an input SQL parameter and an output SQL parameter, A_i shall be assignable to P_i, according to the Syntax Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992, with P_i and A_i as *TARGET* and *VALUE*, respectively.

ii) Otherwise:

1) Let T_i be the data type name that is included in the data type descriptor of the data type of A_i. The *typeset* TS_i of A_i is the set containing a single element T_i.

2) For each TS_i, let VS_i be a set of values such that each element is some value, arbitrarily chosen, in some data type in TS_i and for each data type in TS_i there is exactly one such value in VS_i. Let *XAL* be the set of all <SQL argument list>s XAL_j derived from *AL* by forming the Cartesian product of all value sets VS_i. For each such XAL_j, the Syntax Rules of Subclause 8.1, "Subject routine determination", are applied to the compatible candidate routines of *RI* and XAL_j, yielding a set of candidate subject routines CSR_j for each XAL_j.

Case:

A) If *RN* contains a <schema name>, then for each XAL_j, there shall be exactly one candidate subject routine in CSR_j. The *subject routine* SR_j of XAL_j is the candidate subject routine in CSR_j.

B) Otherwise, for each XAL_j:

I) There shall be at least one candidate subject routine in CSR_j.

II) Case:

1) If there is exactly one candidate subject routine in CSR_j, then the *subject routine* SR_j of XAL_j is the candidate subject routine in CSR_j.

2) If there is more than one candidate subject routine in CSR_j, then the *subject routine* SR_j of XAL_j is an SQL-invoked routine $R1_j$ in CSR_j such that there is no other candidate subject routine $R2_j$ in CSR_j for which the <schema name> of the schema that includes $R2_j$ precedes in *DP* the <schema name> of the schema that includes $R1_j$.

NOTE 9 – The set *XAL* of *RI* is the set of SQL argument lists XAL_j of all possible combinations of data types in *AL*.

NOTE 10 – For each XAL_j there is one subject routine SR_j.

NOTE 11 – There is exactly one XAL_j.

3) The subject routines of *RI* are the SQL-invoked routines contained in the set union of all SR_j.

4) The effective returns data type of *RI* is defined as follows:

A) Let RT_j be the data type returned by SR_j.

B) The effective returns data type of *RI* is the data type resulting from applying the Syntax Rules of Subclause 9.3, "Set operation result data type and nullabilities", in ISO/IEC 9075:1992, to all RT_j.

Access Rules

None.

General Rules

1) Let *SAL* be the static SQL argument list of the <routine invocation>.
NOTE 12 – "static SQL argument list" is defined by the Syntax Rules of this Subclause.

2) Case:

a) If *SAL* is an empty list, then:

i) Let the *dynamic SQL argument list DAL* be *SAL*.

ii) Let *R* be the subject routine of the <routine invocation>.

b) Otherwise:

i) Each SQL argument A_i in *SAL* is evaluated, in any order, to obtain a value V_i.

ii) Let the *dynamic SQL argument list DAL* be the list of values V_i in order.

iii) Let *XAL* be the set of SQL argument lists of the <routine invocation>.

iv) Let XAL_k be the member of *XAL* such that for each SQL argument $A_{i,k}$ in XAL_k the data type of $A_{i,k}$ and the data type of V_i are compatible.

v) Let *R* be the subject routine SR_k of XAL_k.

NOTE 13 – "subject routine" is defined by the Syntax Rules of this Subclause.

3) Let *N* and *PN* be the number of values V_i in *DAL*. Let T_i be the data type of the *i*-th SQL parameter P_i of *R*. For *i* ranging from 1 to *PN*,

Case:

a) If P_i is an input SQL parameter or both an input SQL parameter and an output SQL parameter, then let CPV_i be the result of the assignment of V_i to a target of type T_i according to the rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992.

b) Otherwise, let CPV_i be an implementation-defined value of type T_i.

4) If *R* is a built-in function *BIF*, then

Case:

a) If the syntax for invoking *BIF* is defined in this International Standard, then the result of <routine invocation> is as defined for that syntax in this International Standard.

b) Otherwise, the result of <routine invocation> is implementation-defined.

5) If *R* is an external routine, then let *P* be the program identified by the external name of *R*.

6) Preserve the current SQL-session context *CSC* and create a new SQL-session context *RSC* derived from *CSC* as follows:

a) Set the current default catalog name, the current default unqualified schema name, the current character set name substitution value, the SQL-path of the current SQL-session, the current default time zone, and the contents of all SQL dynamic descriptor areas to implementation-defined values.

b) Set the values of the current SQL-session identifier, the identities of all instances of global temporary tables, the current constraint mode for each integrity constraint, the current transaction access mode, the current transaction isolation level, and the current transaction diagnostics area limit to their values in *CSC*.

c) Case:

i) If *R* is an SQL routine, then remove from *RSC* the identities of all instances of created local temporary tables, declared local temporary tables that are defined by <temporary table declaration>s that are contained in <module>s, and the cursor position of all open cursors.

NOTE 14 – The identities of declared local temporary tables that are defined in <SQL-server module>s are not removed.

ii) Otherwise:

1) Remove from *RSC* the identities of all instances of created local temporary tables that are referenced in <module>s that are not the <module> of *P*, declared local temporary tables that are defined by <temporary table declaration>s that are contained in <module>s that are not the <module> of *P*, and the cursor position of all open cursors that are defined by <declare cursor>s that are contained in <module>s that are not the <module> of *P*.

2) It is implementation-defined whether the identities of all instances of created local temporary tables that are referenced in the <module> of *P*, declared local temporary tables that are defined by <temporary table declaration>s that are contained in the <module> of *P*, and the cursor position of all open cursors that are defined by <declare cursor>s that are contained in the <module> of *P* are removed from *RSC*.

d) Indicate in the routine execution context of *RSC* that the SQL-invoked routine *R* is active.

e) Indicate in the routine execution context of *RSC* whether or not containing SQL, reading SQL-data, or modifying SQL-data is permitted.

NOTE 15 – Such an indication is derived from the routine descriptor of *R*.

f) Case:

i) If *R* is an external routine, then:

1) Set the current <authorization identifier> of *RSC* to be the external routine authorization identifier of *R*.

2) Set the routine SQL-path of *RSC* to be the external routine SQL-path of *R*.

3) If *R* possibly contains SQL, possibly reads SQL, or possibly modifies SQL, then set the SQL-session <module> of *RSC* to be the <module> *M* of *P*; otherwise, set the SQL-session <module> of *RSC* to be an implementation-defined <module>.

ii) Otherwise:

1) Set the current <authorization identifier> of *RSC* to be the routine authorization identifier of *R*.

2) Set the routine SQL-path of *RSC* to be the routine SQL-path of *R*.

3) Set the SQL-session <module> of *RSC* to be the SQL-session <module> of *CSC*.

g) *RSC* becomes the current SQL-session context.

7) Case:

a) If *R* possibly contains SQL and containing SQL is not permitted, then an exception condition is raised: *external routine exception — containing SQL not permitted*.

b) If *R* possibly reads SQL-data and reading SQL-data is not permitted, then:

i) If *R* is an external routine, then an exception condition is raised: *external routine exception — reading SQL-data not permitted*.

ii) Otherwise, an exception condition is raised: *SQL routine exception — reading SQL-data not permitted*.

c) If *R* possibly modifies SQL-data and modifying SQL-data is not permitted, then:

i) If *R* is an external routine, then an exception condition is raised: *external routine exception — modifying SQL-data not permitted*.

ii) Otherwise, an exception condition is raised: *SQL routine exception — modifying SQL-data not permitted*.

d) If *R* does not possibly modify SQL-data, then the routine execution context is set to indicate that modifying SQL-data is not permitted.

e) If *R* does not possibly read SQL-data, then the routine execution context is set to indicate that neither modifying nor reading SQL-data is permitted.

f) If *R* does not possibly contain SQL, then the routine execution context is set to indicate that neither modifying SQL-data, reading SQL-data, nor containing SQL is permitted.

NOTE 16 – Otherwise, the routine execution context is unaltered.

8) If *R* is an SQL routine, then:

a) For *i* ranging from 1 to *PN*, set the value of P_i to CPV_i.

b) The General Rules of Subclause 12.5, "<SQL procedure statement>", in ISO/IEC 9075:1992, are evaluated with the <SQL routine body> of *R* as the *executing statement*.

c) If, before the completion of the execution of the <SQL routine body> of *R*, an attempt is made to execute an SQL-transaction statement or an SQL-connection statement, then an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted*.

d) If reading SQL-data is not permitted and, before the completion of the execution of the <SQL routine body> of *R*, an attempt is made to execute an <SQL procedure statement> that possibly reads SQL-data, then an exception condition is raised: *SQL routine exception — reading SQL-data not permitted*.

e) If modifying SQL-data is not permitted and, before the completion of the execution of the <SQL routine body> of *R*, an attempt is made to execute an SQL-data change statement or an SQL-schema statement, then an exception condition is raised: *SQL routine exception — modifying SQL-data not permitted*.

f) If there is an unhandled exception or completion condition other than *successful completion* at completion of the execution of the <SQL routine body> or *R*, then that condition is re-raised by the <routine invocation>.

g) If *R* is an SQL-invoked function, then:

i) If no <return statement> is executed before completion of the execution of the <SQL routine body> of *R*, then an exception condition is raised: *SQL routine exception — function executed no return statement*.

ii) Otherwise, let *RV* be the returned value of the execution of the <SQL routine body> of *R*.

NOTE 17 – "Returned value" is defined in Subclause 13.2, "<return statement>".

h) If *R* is an SQL-invoked procedure, then for each SQL parameter of *R* that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, set the value of CPV_i to the value of P_i.

9) If *R* is an external routine, then:

a) The method and time of binding of *P* to the schema or <SQL-server module definition> that includes *R* is implementation-defined.

b) If *R* specifies PARAMETER STYLE SQL, then

i) Case:

1) If *R* is an SQL-invoked function, then the effective SQL parameter list *ESPL* of *R* is set as follows:

A) For *i* ranging from 1 to *PN*, the *i*-th entry in *ESPL* is set to CPV_i.

B) For *i* equal to $PN + 1$, the *i*-th entry in *ESPL* is the result data item.

C) For *i* ranging from $(PN + 1) + 1$ to $(PN + 1) + N$, the *i*-th entry in *ESPL* is the SQL indicator argument corresponding to CPV_i.

D) For *i* equal to $(PN + 1) + N + 1$, the *i*-th entry in *ESPL* is the SQL indicator argument corresponding to the result data item.

E) For *i* equal to $(PN + 1) + (N + 1) + 1$, the *i*-th entry in *ESPL* is the exception data item.

F) For *i* equal to $(PN + 1) + (N + 1) + 2$, the *i*-th entry in *ESPL* is the routine name text item.

G) For i equal to $(PN + 1) + (N + 1) + 3$, the i-th entry in *ESPL* is the specific name text item.

H) For i equal to $(PN + 1) + (N + 1) + 4$, the i-th entry in *ESPL* is the message text item.

I) Set the value of the SQL indicator argument corresponding to the result data item (that is, SQL argument value list entry $(PN + 1 + N) + 1$ to 0.

J) For i ranging from 1 to PN, if CPV_i is the null value, then set entry $(PN + 1) + i$ (that is, the i-th SQL indicator argument corresponding to CPV_i) to -1; otherwise, set entry $(PN + 1) + i$ (that is, the i-th SQL indicator argument corresponding to CPV_i) to 0.

2) Otherwise, the effective SQL parameter list *ESPL* of R is set as follows:

A) For i ranging from 1 to PN, the i-th entry in *ESPL* is CPV_i.

B) For i ranging from $PN+1$ to $PN+N$, the i-th entry in *ESPL* is the SQL indicator argument corresponding to CPV_i.

C) For i equal to $(PN + N) + 1$, the i-th entry in *ESPL* is the exception data item.

D) For i equal to $(PN+N)+2$, the i-th entry in *ESPL* is the routine name text item.

E) For i equal to $(PN+N)+3$, the i-th entry in *ESPL* is the specific name text item.

F) For i equal to $(PN + N) + 4$, the i-th entry in *ESPL* is the message text item.

G) For i ranging from 1 to PN, if CPV_i is the null value, then set entry $PN + i$ in *ESPL* (that is, the i-th SQL indicator argument corresponding to CPV_i) to -1; otherwise, set entry $PN + i$ in *ESPL* (that is, the i-th SQL indicator argument corresponding to CPV_i) to 0.

ii) The exception data item is set to '00000'.

iii) The routine name text item is set to the <qualified name> of the routine name of R.

iv) The specific name text item is set to the <qualified identifier> of the specific name of R.

v) The message text item is set to a zero-length string.

c) If the SQL-invoked routine specified PARAMETER STYLE GENERAL, then the effective SQL parameter list *ESPL* of R is set as follows:

i) For i ranging from 1 to PN, if any CPV_i is the null value, then an exception condition is raised: *external routine invocation exception — null value not allowed*.

ii) For i ranging from 1 to PN, the i-th entry in *ESPL* is set to CPV_i.

d) If R specifies DETERMINISTIC and if different executions of P with identical SQL argument value lists do not produce identical results, then the results are implementation-dependent.

e) Let *EN* be the number of entries in *ESPL*. Let ESP_i be the i-th effective SQL parameter in *ESPL*.

f) *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

i) Depending on whether the language of *R* specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences table* be Table 3, "Data type correspondences for Ada", Table 4, "Data type correspondences for C", Table 5, "Data type correspondences for COBOL", Table 6, "Data type correspondences for Fortran", Table 7, "Data type correspondences for MUMPS", Table 8, "Data type correspondences for Pascal", or Table 9, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the "SQL data type" column and the "host data type" column.

ii) For *i* varying from 1 to *EN*, the <data type> DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

iii) The value of PD_i is set to the value of ESP_i.

g) If, before the completion of the execution of *P*, an attempt is made to execute an SQL-transaction statement or an SQL-connection statement, then an exception condition is raised: *external routine exception — prohibited SQL-statement attempted*.

h) If containing SQL is not permitted and, before the completion of the execution of *P*, an attempt is made to execute an <SQL procedure statement> that possibly contains SQL, then an exception condition is raised: *external routine exception — containing SQL not permitted*.

i) If reading SQL-data is not permitted and, before the completion of the execution of *P*, an attempt is made to execute an <SQL procedure statement> that possibly reads SQL-data, then an exception condition is raised: *external routine exception — reading SQL-data not permitted*.

j) If modifying SQL-data is not permitted and, before the completion of the execution of *P*, an attempt is made to execute an SQL-data change statement or an SQL-schema statement, then an exception condition is raised: *external routine exception — modifying SQL-data not permitted*.

k) If the language specifies ADA (respectively C, COBOL, FORTRAN, MUMPS, PASCAL, PLI) and *P* is not a standard-conforming Ada program (respectively C, COBOL, Fortran, MUMPS, Pascal, PL/I program), then the results of the execution of *P* are implementation-dependent.

l) After the completion of the execution of *P*:

i) It is implementation-defined whether:

1) For every open cursor *CR* that is associated with *RSC* and that is defined by a <declare cursor> that is contained in the <module> of *P*:

A) The following SQL-statement is effectively executed:

```
CLOSE CR
```

B) *CR* is destroyed.

2) Every instance of created local temporary tables and every instance of declared local temporary tables that is associated with *RSC* is destroyed.

3) For every prepared statement *PS* prepared by *P* in the current SQL-transaction that has not been deallocated by *P*:

A) Let *SSN* be the <SQL statement name> that identifies *PS*.

B) The following SQL-statement is effectively executed:

```
DEALLOCATE PREPARE SSN
```

ii) For *i* varying from 1 to *EN*, the value of ESP_i is set to the value of PD_i.

Case:

1) If the exception data item has the value '00000', then the execution of *P* was successful.

2) If the first two characters of the exception data item are equal to the SQLSTATE exception code class value for *external routine exception*, then an exception condition is raised: *external routine exception*, using a subclass code equal to the final three characters of the value of the exception data item.

3) If the first two characters of the exception data item are equal to the SQLSTATE exception code class value for *warning* and the third character of the exception data item is 'H', then a completion condition is raised: *warning*, using a subclass code equal to the final three characters of the value of the exception data item.

4) Otherwise, an exception condition is raised: *external routine invocation exception — invalid SQLSTATE returned*.

iii) If the exception data item is not '00000' and *R* specified PARAMETER STYLE SQL, then the message text item is stored in the diagnostics area.

m) If *R* is an SQL-invoked function, then:

i) Case:

1) If *R* specified PARAMETER STYLE SQL and if entry $(PN+1)+N+1$ in *ESPL* (that is, SQL indicator argument $N+1$ corresponding to the result data item) is negative, then let *RDI* be the null value; otherwise, let *RDI* be the value of the result data item.

2) Otherwise, let *RDI* be the value returned from *P*. The value returned from *P* is passed to the SQL-implementation in an implementation-dependent manner. An argument value list entry is not used for this purpose.

ii) If *R* specified a <result cast>, then let *RT* be the <returns data type> of *R* and let *RV* be the result of:

```
CAST ( RDI AS RT )
```

Otherwise, let *RV* be *RDI*.

n) If *R* is an SQL-invoked procedure, then for each SQL parameter of *R* that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, then

Case:

i) If *R* specifies PARAMETER STYLE SQL, then for *i* ranging from 1 to *PN*,

Case:

1) If entry $(PN+1)+i$ in *ESPL* (that is, the *i*-th SQL indicator argument corresponding to CPV_i) is negative, then CPV_i is set to the null value.

2) If a value was not assigned to the *i*-th entry in *ESPL*, then CPV_i is set to an implementation-defined value of type T_i.

3) Otherwise, CPV_i is set to the value of the *i*-th entry in *ESPL*.

ii) Otherwise, for *i* ranging from 1 to *PN*, CPV_i is set to the value of the *i*-th entry in *ESPL*.

10) Case:

a) If *R* is an SQL-invoked function, then:

i) Let *ERDT* be the effective returns data type of the <routine invocation>.

ii) Let the result of the <routine invocation> be the result of assigning *RV* to a target of type *ERDT* according to the rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992.

b) Otherwise, for each SQL parameter P_i of *R* that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, let TS_i be the <target specification> of the corresponding <SQL argument> A_i; CPV_i is assigned to TS_i according to the rules of Subclause 9.1, "Retrieval assignment", in ISO/IEC 9075:1992.

11) Prepare *CSC* to become the current SQL-session context:

a) Set the value of the current constraint mode for each integrity constraint in *CSC* to the value of the current constraint mode for each integrity constraint in *RSC*.

b) Set the value of the current transaction access mode in *CSC* to the value of the current transaction access mode in *RSC*.

c) Set the value of the current transaction isolation level in *CSC* to the value of the current transaction isolation level in *RSC*.

d) Set the value of the current transaction diagnostics area limit in *CSC* to the value of the current transaction diagnostics area limit in *RSC*.

e) Replace the identities of all instances of global temporary tables in *CSC* with the identities of the instances of global temporary tables in *RSC*.

12) *CSC* becomes the current SQL-session context.

9.2 <privileges>

Function

Specify privileges.

Format

```
<action> ::=
         !! All alternatives from ISO/IEC 9075:1992
       | EXECUTE
```

Syntax Rules

1) Insert this SR If the object identified by <object name> of the <grant statement> or <revoke statement> is a schema-level routine or an SQL-server module, then <privileges> shall specify EXECUTE; otherwise, EXECUTE shall not be specified.

   NOTE 18 – "schema-level routine" is defined in Subclause 10.18, "<SQL-invoked routine>".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

9.3 <specific routine designator>

Function

Specify an SQL-invoked routine.

Format

```
<specific routine designator> ::=
         SPECIFIC <routine type> <specific name>
       | <routine type> <member name>

<routine type> ::=
         ROUTINE
       | FUNCTION
       | PROCEDURE

<member name> ::= <routine name> [ <data type list> ]

<data type list> ::=
       <left paren> [ <data type> [ { <comma> <data type> }... ] ] <right paren>
```

Syntax Rules

1) If a <specific name> *SN* is specified, then the <specific routine designator> identifies the SQL-invoked routine whose <specific name> is *SN*.

2) If a <member name> *MN* is specified, then:

   a) Let *RN* be the <routine name> of *MN* and let *SN* be the <schema name> of *MN*.

   b) Case:

      i) If *MN* contains a <data type list>, then:

         1) If <routine type> specifies FUNCTION, then there shall be exactly one SQL-invoked function in the schema identified by *SN* whose <routine name> is *RN* such that for all *i* the data type of its *i*-th SQL parameter is identical to the *i*-th <data type> in the <data type list> of *MN*. The <specific routine designator> identifies that SQL-invoked function.

         2) If <routine type> specifies PROCEDURE, then there shall be exactly one SQL-invoked procedure in the schema identified by *SN* whose <routine name> is *RN* such that for all *i* the data type of its *i*-th SQL parameter is identical to the *i*-th <data type> in the <data type list> of *MN*. The <specific routine designator> identifies that SQL-invoked function.

         3) If <routine type> specifies ROUTINE, then there shall be exactly one SQL-invoked routine in the schema identified by *SN* whose <routine name> is *RN* such that for all *i* the data type of its *i*-th SQL parameter is identical to the *i*-th <data type> in the <data type list> of *MN*. The <specific routine designator> identifies that SQL-invoked routine.

ii) Otherwise:

1) If <routine type> specifies FUNCTION, then there shall be exactly one SQL-invoked function in the schema identified by *SN* whose <routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked function.

2) If <routine type> specifies PROCEDURE, then there shall be exactly one SQL-invoked procedure in the schema identified by *SN* whose <routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked procedure.

3) If <routine type> specifies ROUTINE, then there shall be exactly one SQL-invoked routine in the schema identified by *SN* whose <routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked routine.

3) If FUNCTION is specified, then the SQL-invoked routine that is identified shall be an SQL-invoked function. If PROCEDURE is specified, then the SQL-invoked routine that is identified shall be an SQL-invoked procedure.

Access Rules

None.

General Rules

None.

9.4 <sqlstate value>

Function

Specify an SQLSTATE value.

Format

```
<sqlstate value> ::=
       SQLSTATE [ VALUE ] <character string literal>
```

Syntax Rules

1) Let *L* be the character string that is the value of the <character string literal> contained in <sqlstate value>.

2) The implicit or explicit character set of *L* shall be the implementation-defined character set in which SQLSTATE parameter values are returned.

3) Let *V* be the character string that is the value of

   ```
   TRIM ( BOTH ' ' FROM L )
   ```

4) The value of *V* shall comprise either:

   a) A standard SQLSTATE Class and Subclass value;

   b) A standard SQLSTATE Class value for which an implementation-defined Subclass value is permitted and three characters with the form of an implementation-defined Subclass value; or

   c) Five characters of which the first two have the form of an implementation-defined Class value.

5) The value of *V* shall not be the SQLSTATE value for the condition *successful completion*.

6) The SQLSTATE value defined by the <sqlstate value> is the value of *V*.

Access Rules

None.

General Rules

None.

9.5 <language clause>

Function

Specify a standard programming language.

Format

```
<language name> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | SQL
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

Table 2—Standard programming languages

Language keyword	Relevant standard
All alternatives from ISO/IEC 9075:1992	
SQL	ISO/IEC 9075

9.6 <path specification>

Function

Specify an order for searching for an SQL-invoked routine.

Format

```
<path specification> ::=
       PATH <schema name list>

<schema name list> ::=
       <schema name> [ { <comma> <schema name> }... ]
```

Syntax Rules

1) No <schema name> shall appear more than once in <schema name list>.

2) If no <schema name> in <schema name list> *SO* contains the <qualified identifier> INFORMATION_SCHEMA, then *SO* is effectively replaced by

   INFORMATION_SCHEMA, *SO*

Access Rules

None.

General Rules

None.

10 Schema definition and manipulation

10.1 <schema definition>

Function

Define a schema.

Format

```
<schema definition> ::=
      CREATE SCHEMA <schema name clause>
        [ <schema character set specification> ]
        [ <schema path specification> ]
        [ <schema element>... ]

<schema path specification> ::=
      <path specification>

<schema element> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <SQL-server module definition>
      | <SQL-invoked routine>
```

Syntax Rules

1) Insert this SR The SQL-invoked routine specified by <SQL-invoked routine> shall be a schema-level routine.

   NOTE 19 – "Schema-level routine" is defined in Subclause 10.18, "<SQL-invoked routine>".

2) Insert this SR If <schema path specification> is not specified, then a <schema path specification> containing an implementation-defined <schema name list> that includes the <schema name> contained in <schema name clause> is implicit.

3) Insert this SR The explicit or implicit <catalog name> of each <schema name> contained in the <schema name list> of the <schema path specification> shall be equivalent to the <catalog name> of the <schema name> contained in <schema name clause>.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR The <schema name list> of the explicit or implicit <schema path specification> is used as the SQL-path of the schema. The SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <schema definition>.

2) Replace GR3) Those objects defined by <schema element>s (base tables, views, constraints, domains, assertions, character sets, translations, collations, privileges, SQL-server modules, SQL-invoked routines) and their associated descriptors are effectively created.

10.2 <drop schema statement>

Function

Destroy a schema.

Format

No additional Format items.

Syntax Rules

1) Replace SR3) If RESTRICT is specified, then *S* shall not contain any persistent base tables, global temporary tables, created local temporary tables, views, domains, assertions, character sets, collations, translations, SQL-server modules, or SQL-invoked routines, and the <schema name> of *S* shall not be contained in the <SQL routine body> of any routine descriptor or in the SQL-path of any SQL-server module descriptor.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR8) For every SQL-server module *M* contained in *S*, let *MN* be the <SQL-server module name> of *M*. For every *M*, the following <drop module statement> is effectively executed:

```
DROP MODULE MN CASCADE
```

2) Insert before GR8) For every SQL-invoked routine *R* contained in *S*, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed for every *R*:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

3) Insert before GR8) Let *R* be any SQL-invoked routine whose routine descriptor contains the <schema name> of *S* in the <SQL routine body>.

Case:

a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

4) Insert before GR8) Let *SSM* be any SQL-server module whose module descriptor includes the <schema name> of *S* and let *MN* be the <SQL-server module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

10.3 <default clause>

Function

Specify the default for a column, domain, or SQL variable.

Format

```
<default option> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | CURRENT_PATH
```

Syntax Rules

1) Replace SR1) The subject data type of a <default clause> is the data type specified in the descriptor identified by the containing <column definition>, <domain definition>, <alter column definition>, or <alter domain statement>, or that defined by the <data type> specified in the containing <SQL variable declaration>.

2) Replace SR3)b) If CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH is specified, then the subject data type shall be character string with character set SQL_TEXT. If the length of the subject data type is fixed, then its length shall not be less than 128 characters. If the length of the subject data type is variable, then its maximum length shall not be less than 128 characters.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR2) in the TC

NOTE 20 – If <default option> specifies CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, then the "value in the column descriptor" will effectively be the text of the <default option>, whose evaluation occurs at the time that the default value is required.

2) Replace GR2.1)c) in the TC If the column or domain descriptor specifies CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, then

Case:

i) If the subject data type is character string with variable length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, respectively, at the time that the default value is required.

ii) If the subject data type is character string with fixed length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, respectively, at the time that the default value is required, extended as necessary on the right with <space>s to the length in characters of the subject data type.

3) Insert after GR2.1) in the TC The default value of an SQL variable is

Case:

a) If the <SQL variable declaration> contains a <default clause>, then the value specified by that <default clause>.

b) Otherwise, the null value.

4) Insert after GR2.1) in the TC When a default value is required for an SQL variable, the default value for the variable is derived from the <default option> as follows:

a) If the <default option> specifies NULL, then the null value.

b) If the <default option> contains a <literal>, then

Case:

i) If the subject data type is numeric, then the numeric value of the <literal>.

ii) If the subject data type is character string with variable length, then the value of the <literal>.

iii) If the subject data type is character string with fixed length, then the value of the <literal>, extended as necessary on the right with <space>s to the length in characters of the subject data type.

iv) If the subject data type is bit string with variable length, then the value of the <literal>.

v) If the subject data type is datetime or interval, then the value of the <literal>.

c) If the <default option> specifies CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, then

Case:

i) If the subject data type is character string with variable length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, respectively, at the time that the default value is required.

ii) If the subject data type is character string with fixed length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, respectively, at the time that the default value is required, extended as necessary on the right with <space>s to the length in characters of the subject data type.

d) If the <default option> contains a <datetime value function>, then the value of an evaluation of the <datetime value function> at the time that the default value is required.

10.4 <check constraint definition>

Function

Specify a condition for the SQL-data.

Format

No additional Format items.

Syntax Rules

1) Insert following SR1)
NOTE 21 – <SQL parameter name> and <SQL variable name> are also excluded because of the scoping rules for <SQL parameter name> and <SQL variable name>.

2) Insert this SR The <search condition> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic.

3) Insert this SR The <search condition> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

4) Insert this SR The <search condition> shall not generally contain a <value specification> that is CURRENT_PATH.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

10.5 <drop column definition>

Function

Destroy a column.

Format

No additional Format items.

Syntax Rules

1) Replace SR4) If RESTRICT is specified, then *C* shall not be referenced in any of the following:

   a) The <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor other than a table constraint descriptor that contains references to no other column and that is included in the table descriptor of *T*,

   c) The <SQL routine body> of any SQL-invoked routine descriptor.

   d) The module descriptor of any SQL-server module.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR3) Let *R* be any SQL-invoked routine whose routine descriptor contains the <column name> of *C* in the <SQL routine body>.

   Case:

   a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

      DROP MODULE *MN* CASCADE

   b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

      DROP SPECIFIC ROUTINE *SN* CASCADE

2) Insert before GR3) Let *SSM* be any SQL-server module whose module descriptor contains the <column name> of *C* and let *MN* be the <SQL-server module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

   DROP MODULE *MN* CASCADE

10.6 <drop table constraint definition>

Function

Destroy a constraint on a table.

Format

No additional Format items.

Syntax Rules

1) Replace SR4) If RESTRICT is specified, then no table constraint shall be dependent on *TC* and the <constraint name> of *TC* shall not be contained in the <SQL routine body> of any routine descriptor.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR2) Let *R* be any SQL-invoked routine whose routine descriptor contains the <constraint name> of *TC* in the <SQL routine body>.

Case:

a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

10.7 <drop table statement>

Function

Destroy a table.

Format

No additional Format items.

Syntax Rules

1) Replace SR4) If RESTRICT is specified, then *T* shall not be referenced in any of the following:

   a) The <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor.

   c) The <SQL routine body> of any routine descriptor.

   d) The module descriptor of any SQL-server module.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR2) Let *R* be any SQL-invoked routine whose routine descriptor contains the <table name> of *T* in the <SQL routine body>.

   Case:

   a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

      DROP MODULE *MN* CASCADE

   b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

      DROP SPECIFIC ROUTINE *SN* CASCADE

2) Insert before GR2) Let *SSM* be any SQL-server module whose module descriptor contains the <table name> of *T* and let *MN* be the <SQL-server module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

   DROP MODULE *MN* CASCADE

10.8 <view definition>

Function

Define a viewed table.

Format

No additional Format items.

Syntax Rules

1) Insert following SR1)

NOTE 22 – <SQL parameter name> and <SQL variable name> are also excluded because of the scoping rules for <SQL parameter name> and <SQL variable name>.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR4) A set of privilege descriptors is created that defines the privilege SELECT on this table to *A* and SELECT for each column of *V* to *A*. This privilege is grantable if and only if the applicable SELECT privileges on all <table name>s contained in the <query expression> are grantable and the applicable EXECUTE privileges on all SQL-invoked routines that are subject routines of a <routine invocation> contained in the <query expression> are grantable. The grantor of this privilege descriptor is set to the special grantor value "_SYSTEM".

10.9 <drop view statement>

Function

Destroy a view.

Format

No additional Format items.

Syntax Rules

1) Replace SR3) If RESTRICT is specified, then *V* shall not be referenced in any of the following:

   a) The <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor.

   c) The <SQL routine body> of any routine descriptor.

   d) The module descriptor of any SQL-server module.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR2) Let *R* be any SQL-invoked routine whose routine descriptor contains the <table name> of *V* in the <SQL routine body>.

   Case:

   a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

      DROP MODULE *MN* CASCADE

   b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

      DROP SPECIFIC ROUTINE *SN* CASCADE

2) Insert before GR2) Let *SSM* be any SQL-server module whose module descriptor contains the <table name> of *V* and let *MN* be the <module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

   DROP MODULE *MN* CASCADE

10.10 <drop domain statement>

Function

Destroy a domain.

Format

No additional Format items.

Syntax Rules

1) Replace SR2) If RESTRICT is specified, then *D* shall not be referenced in any of the following:

   a) The column descriptor of any column.

   b) The <query expression> of any view descriptor.

   c) The <search condition> of any constraint descriptor.

   d) The module descriptor of any SQL-server module.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR3) Let *SSM* be any SQL-server module whose module descriptor contains the <domain name> of *D* and let *MN* be the <SQL-server module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

10.11 <drop character set statement>

Function

Destroy a character set.

Format

No additional Format items.

Syntax Rules

1) Replace SR3) *C* shall not be referenced in any of the following:

   a) The <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor.

   c) The collation descriptor of any collation.

   d) The translation descriptor of any translation.

   e) The <SQL routine body> or the <SQL parameter declaration>s of any routine descriptor.

   f) The module descriptor of any SQL-server module.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR2) Let *R* be any SQL-invoked routine whose routine descriptor contains the <character set name> of *C* in the <SQL routine body> or the <SQL parameter declaration>s.

   Case:

   a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

      DROP MODULE *MN* CASCADE

   b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

      DROP SPECIFIC ROUTINE *SN* CASCADE

2) Insert before GR2) Let *SSM* be any SQL-server module whose module descriptor contains the <character set name> of *C* and let *MN* be the <SQL-server module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

   DROP MODULE *MN* CASCADE

10.12 <drop collation statement>

Function

Destroy a collating sequence.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR2) Let *SSM* be any SQL-server module whose module descriptor contains the <collation name> of *C* and let *MN* be the <SQL-server module name> of *SSM*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

2) Insert before GR6) Let *R* be any SQL-invoked routine whose routine descriptor contains the <collation name> of *C* in the <SQL routine body> or the <SQL parameter declaration>s.

Case:

a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

10.13 <drop translation statement>

Function

Destroy a character translation.

Format

No additional Format items.

Syntax Rules

1) Replace SR3) T shall not be referenced in any of the following:

   a) The <query expression> included in any view descriptor.

   b) The <search condition> included in any constraint descriptor.

   c) The collation descriptor of any collation.

   d) The <SQL routine body> of any routine descriptor.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR4) Let R be any SQL-invoked routine whose routine descriptor contains the <translation name> of T in the <SQL routine body> or the <SQL parameter declaration>s.

   Case:

   a) If R is included in an SQL-server module M, then let MN be the <SQL-server module name> of M. The following <drop module statement> is effectively executed without further Access Rule checking:

      DROP MODULE *MN* CASCADE

   b) Otherwise, let SN be the <specific name> of R. The following <drop routine statement> is effectively executed without further Access Rule checking:

      DROP SPECIFIC ROUTINE *SN* CASCADE

10.14 <assertion definition>

Function

Specify an integrity constraint by means of an assertion and specify the initial default time for checking the assertion.

Format

No additional Format items.

Syntax Rules

1) Insert after SR4)
NOTE 23 – <SQL parameter name> and <SQL variable name> are also excluded because of the scoping rules for <SQL parameter name> and <SQL variable name>.

2) Insert this SR The <search condition> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic.

3) Insert this SR The <search condition> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

4) Insert this SR The <search condition> shall not generally contain a <value specification> that is CURRENT_PATH.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

10.15 <drop assertion statement>

Function

Destroy an assertion.

Format

No additional Format items.

Syntax Rules

1) Insert this SR The <constraint name> of *AN* shall not be referenced in the <SQL routine body> of any routine descriptor.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR1) Let *R* be any SQL-invoked routine whose routine descriptor contains the <constraint name> of *A* in the <SQL routine body>.

Case:

a) If *R* is included in an SQL-server module *M*, then let *MN* be the <SQL-server module name> of *M*. The following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

b) Otherwise, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

10.16 <SQL-server module definition>

Function

Define an SQL-server module.

Format

```
<SQL-server module definition> ::=
       CREATE MODULE <SQL-server module name>
         [ <SQL-server module character set specification> ]
         [ <SQL-server module schema clause> ]
         [ <SQL-server module path specification> ]
         [ <temporary table declaration> ]
         <SQL-server module contents>...
       END MODULE

<SQL-server module character set specification> ::=
       NAMES ARE <character set specification>

<SQL-server module schema clause> ::=
       SCHEMA <default schema name>

<default schema name> ::=
       <schema name>

<SQL-server module path specification> ::=
       <path specification>

<SQL-server module contents> ::=
         <SQL-invoked routine> <semicolon>
```

Syntax Rules

1) If an <SQL-server module definition> is contained in a <schema definition> *SD* and the <SQL-server module name> of the <SQL-server module definition> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of *SD*.

2) The schema identified by the explicit or implicit <schema name> of the <SQL-server module name> shall not include a module descriptor whose <SQL-server module name> is equivalent to the <SQL-server module name> of the containing <SQL-server module definition>.

3) The SQL-invoked routine specified by <SQL-invoked routine> shall not be a schema-level routine.

   NOTE 24 – "Schema-level routine" is defined in Subclause 10.18, "<SQL-invoked routine>".

4) If <SQL-server module path specification> is not specified, then an <SQL-server module path specification> containing an implementation-defined <schema name list> that includes the explicit or implicit <schema name> of the <SQL-server module name> is implicit.

5) The explicit or implicit <catalog name> of each <schema name> contained in the <schema name list> of the <SQL-server module path specification> shall be equivalent to the <catalog name> of the explicit or implicit <schema name> of the <SQL-server module name>.

6) The <schema name list> of the explicit or implicit <SQL-server module path specification> is used as the SQL-path of the SQL-server module. The SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <SQL-server module definition>.

7) If <SQL-server module schema clause> is not specified, then an <SQL-server module schema clause> containing the <default schema name> that is equivalent to the explicit or implicit <schema name> of the <SQL-server module name> is implicit.

8) If <SQL-server module character set specification> is not specified, then an <SQL-server module character set specification> containing the <character set specification> that is equivalent to the <schema character set specification> of the schema identified by the explicit or implicit <schema name> of the <SQL-server module name> is implicit.

Access Rules

1) If an <SQL-server module definition> is contained in a <module> with no intervening <schema definition>, then the current <authorization identifier> shall be equivalent to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <SQL-server module name>.

General Rules

1) An <SQL-server module definition> defines an SQL-server module.

2) A privilege descriptor is created that defines the EXECUTE privilege on the SQL-server module to the <authorization identifier> that owns the schema identified by the explicit or implicit <schema name> of the <SQL-server module name>. The grantor for the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable if and only if all of the privileges necessary for the <authorization identifier> to successfully execute the <SQL procedure statement> contained in the <routine body> of every <SQL-invoked routine> contained in the <SQL-server module definition> are grantable.

   NOTE 25 – The necessary privileges include the EXECUTE privilege on every subject routine of every <routine invocation> contained in the <SQL procedure statement>.

3) An SQL-server module descriptor is created that describes the SQL-server module being defined. The SQL-server module descriptor includes:

   a) The SQL-server module name specified by the <SQL-server module name>.

   b) The descriptor of the character set specified by the <SQL-server module character set specification>.

   c) The default schema name specified by the <SQL-server default schema clause>.

   d) The SQL-server module authorization identifier that corresponds to the authorization identifier that owns the schema identified by the explicit or implicit <schema name> of the <SQL-server module name>.

   e) The list of schema names contained in the <SQL-server module path specification>.

   f) The descriptor of every local temporary table declared in the SQL-server module.

   g) The descriptor of every SQL-invoked routine contained in the SQL-server module.

h) The text of the <SQL-server module definition>.

10.17 <drop module statement>

Function

Destroy an SQL-server module.

Format

```
<drop module statement> ::=
       DROP MODULE <SQL-server module name> <drop behavior>
```

Syntax Rules

1) Let *MN* be the <SQL-server module name> and let *M* be the SQL-server module identified by *MN*.

2) *M* shall be an SQL-server module.

3) If RESTRICT is specified, then *M* shall not contain an SQL-invoked routine that is included in the subject routines of a <routine invocation> that is contained in any of the following:

   a) The <SQL routine body> of any routine descriptor not included in the module descriptor of *M*.

   b) The <query expression> of any view descriptor.

   c) The <search condition> of any constraint descriptor or assertion descriptor.

   d) The module descriptor of any SQL-server module other than *M*.

Access Rules

1) The current <authorization identifier> shall be equivalent to the <authorization identifier> that owns the schema identified by the <schema name> of *M*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   ```
   REVOKE EXECUTE ON MODULE MN FROM A CASCADE
   ```

2) *M* and its descriptor are destroyed.

10.18 <SQL-invoked routine>

Function

Define an SQL-invoked routine.

Format

```
<SQL-invoked routine> ::=
          <module routine>
        | <schema routine>

<module routine> ::=
          <module procedure>
        | <module function>

<schema routine> ::=
          <schema procedure>
        | <schema function>

<module procedure> ::=
        [ DECLARE ] <SQL-invoked procedure>

<SQL-invoked procedure> ::=
        PROCEDURE <routine name>
          <SQL parameter list>
          [ <language clause> ]
          [ SPECIFIC <specific name> ]
          [ <deterministic attribute> ]
          [ <SQL-data access indication> ]
          <routine body>

<module function> ::=
        [ DECLARE ] <SQL-invoked function>

<SQL-invoked function> ::=
        FUNCTION <routine name>
          <SQL parameter list>
          <returns clause>
          [ <language clause> ]
          [ SPECIFIC <specific name> ]
          [ <deterministic attribute> ]
          [ <SQL-data access indication> ]
          <routine body>

<schema procedure> ::=
        CREATE <SQL-invoked procedure>

<schema function> ::=
        CREATE <SQL-invoked function>

<SQL parameter list> ::=
        <left paren>
          [ <SQL parameter declaration> [ { <comma> <SQL parameter declaration> }... ] ]
        <right paren>

<SQL parameter declaration> ::=
          [ <parameter mode> ] [ <SQL parameter name> ]
          <parameter type>
```

```
<parameter mode> ::=
        IN
      | OUT
      | INOUT

<parameter type> ::=
      <data type>

<returns clause> ::= RETURNS <returns data type> [ <result cast> ]

<result cast> ::= CAST FROM <data type>

<returns data type> ::= <data type>

<routine body> ::=
        <SQL routine body>
      | <external body reference>

<SQL routine body> ::= <SQL procedure statement>

<external body reference> ::=
      EXTERNAL [ NAME <external routine name> ]
      [ <parameter style> ]

<parameter style> ::=
        PARAMETER STYLE SQL
      | PARAMETER STYLE GENERAL

<deterministic attribute> ::=
        DETERMINISTIC
      | NOT DETERMINISTIC

<SQL-data access indication> ::=
        NO SQL
      | CONTAINS SQL
      | READS SQL DATA
      | MODIFIES SQL DATA
```

Syntax Rules

1) An <SQL-invoked routine> specifies an *SQL-invoked routine*.

2) If <SQL-invoked routine> immediately contains <schema routine>, then the SQL-invoked routine identified by <routine name> is a *schema-level routine*.

3) An SQL-invoked routine specified as an <SQL-invoked procedure> is called an *SQL-invoked procedure*; an SQL-invoked routine specified as an <SQL-invoked function> is called an *SQL-invoked function*.

4) If <language clause> is not specified, then LANGUAGE SQL is implicit.

5) An SQL-invoked routine that specifies LANGUAGE SQL is called an *SQL routine*; an SQL-invoked routine that specifies a <language clause> that does not specify LANGUAGE SQL is called an *external routine*.

6) The scope of the <routine name> of an <SQL-invoked routine> *R* is the <routine body> of *R*.

7) Let *R* be the SQL-invoked routine specified by <SQL-invoked routine> and let *RN* be the <routine name> of *R*.

8) The <SQL parameter list> *PL* of *R* specifies the list of *SQL parameters* of *R*. Each SQL parameter of *R* is specified by an <SQL parameter declaration>. If <SQL parameter name> is specified, then that SQL parameter of *R* is identified by an *SQL parameter name*.

9) Let *N* and *PN* be the number of SQL parameters of *R*. For *i* ranging from 1 to *PN*, let P_i be the *i*-th SQL parameter of *R*.

10) No two SQL parameters of *R* shall have equivalent <SQL parameter name>s.

11) If <SQL-invoked routine> is contained in a <schema definition> without an intervening <SQL-server module definition> and if *RN* contains a <schema name>, then that <schema name> shall be equivalent to the specified <schema name> of the containing <schema definition>.

12) If <SQL-invoked routine> is contained in an <SQL-server module definition> and if *RN* contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the containing <SQL-server module definition>.

13) Let *S* be the schema identified by the <schema name> of *RN*.

14) The scope of an <SQL parameter name> contained in *PL* is the <routine body> *RB* of the <SQL-invoked procedure> or <SQL-invoked function> that contains *PL*.

15) An SQL-invoked routine shall not contain a <parameter name>, a <dynamic parameter specification>, or an <embedded variable name>.

16) Case:

    a) If *R* is an SQL-invoked procedure, then *S* shall not include another SQL-invoked procedure whose <routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

    b) Otherwise, let *SCR* be the set containing every SQL-invoked function in *S*, including *R*, whose <routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

        i) Let *AL* be an <SQL argument list> constructed from a list of arbitrarily-selected values in which the data type of every value A_i in *AL* is compatible with the data type of the corresponding SQL parameter P_i.

        ii) For every A_i, eliminate from *SCR* every SQL-invoked routine for which the data type of P_i is not in the type precedence list of the data type of A_i.

        iii) Let *SR* be the set of subject routines defined by applying the Syntax Rules of Subclause 8.1, "Subject routine determination", with the set of SQL-invoked routines as *SCR* and <SQL argument list> as *AL*. There shall be exactly one subject routine in *SR*.

17) If <specific name> is not specified, then an implementation-dependent <specific name> whose <schema name> is the same as the <schema name> of *S* is implicit. This implementation-dependent <specific name> shall be different from the <specific name> of any other routine descriptor in *S*.

18) If <specific name> contains a <schema name>, then that <schema name> shall be equivalent to the <schema name> of *S*. If <specific name> does not contain a <schema name>, then the <schema name> of *S* is implicit.

19) The schema identified by the explicit or implicit <schema name> of the <specific name> shall not include a routine descriptor whose specific name is equivalent to <specific name>.

20) If <deterministic attribute> is not specified, then NOT DETERMINISTIC is implicit.

21) If <SQL-data access indication> is not specified, then CONTAINS SQL is implicit.

22) If *R* is an SQL-invoked function, then <parameter mode> shall not be specified.

23) If *R* is an SQL routine, then:

a) <SQL routine body> shall be specified.

b) Each <SQL parameter declaration> in the <SQL parameter list> shall contain an <SQL parameter name>.

c) Whether an <SQL parameter declaration> is for an *input SQL parameter*, an *output SQL parameter*, or both is determined as follows:

i) For every <SQL parameter declaration> for which <parameter mode> is not specified, a <parameter mode> that specifies IN is implicit.

ii) Case:

1) If the <parameter mode> specifies IN, then the SQL parameter is an *input SQL parameter*. The <SQL parameter name> shall not be contained in a <target specification> or a <simple target specification> that is contained in <SQL routine body>.

2) If the <parameter mode> specifies OUT, then the SQL parameter is an *output SQL parameter*. The <SQL parameter name> shall not be contained in a <value specification>, a <simple value specification>, or a <value expression> that is contained in <SQL routine body>.

3) If the <parameter mode> specifies INOUT, then the SQL parameter is both an *input SQL parameter* and an *output SQL parameter*.

d) The <returns clause> shall not specify a <result cast>.

e) <SQL-data access indication> shall not specify NO SQL.

f) The <SQL routine body> shall not contain an <SQL dynamic statement>.

24) If *R* is an external routine, then:

a) <SQL routine body> shall not be specified.

b) If an <external routine name> is not specified, then an <external routine name> that is equivalent to the <qualified identifier> of *R* is implicit.

c) If <parameter style> is not specified, then PARAMETER STYLE SQL is implicit.

d) If a <deterministic attribute> is not specified, then NOT DETERMINISTIC is implicit.

e) If a <result cast> is specified, then let *V* be some value of the <data type> specified in the <result cast> and let *RT* be the <returns data type>. The following shall be valid according to the Syntax Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992:

```
CAST ( V AS RT )
```

f) For every <SQL parameter declaration>, if <parameter mode> is not specified, then IN is implicit.

g) For every <SQL parameter declaration>: If <parameter mode> specifies IN, then the SQL parameter is an *input SQL parameter*. If the <parameter mode> specifies OUT, then the SQL parameter is an *output SQL parameter*. If <parameter mode> specifies INOUT, then the SQL parameter is both an *input SQL parameter* and an *output SQL parameter*.

h) If PARAMETER STYLE SQL is specified, then:

Case:

i) If *R* is an SQL-invoked function, then let *RN* be 1. Let the *effective SQL parameter list* be a list of $PN + RN + N + 5$ SQL parameters, as follows:

1) For i ranging from 1 to *PN*, the i-th effective SQL parameter type list entry consists of the i-th <SQL parameter declaration>.

2) Effective SQL parameter type list entry $PN + RN$ has <parameter mode> OUT and has the <returns data type>.

3) Effective SQL parameter list entries $(PN + RN) + 1$ to $(PN + RN) + N + 1$ are $N + 1$ occurrences of SQL parameters of an implementation-defined <data type> that is an exact numeric type with scale 0. For i ranging from $(PN + RN) + 1$ to $(PN + RN) + N + 1$, the <parameter mode> for the i-th such effective SQL parameter is the same as that of the $i - RN - PN$-th effective SQL parameter.

4) Effective SQL parameter type list entry $(PN + RN) + (N + 1) + 1$ is an SQL parameter of a <data type> that is character string of length 5 and character set SQL_TEXT with <parameter mode> INOUT.

5) Effective SQL parameter type list entry $(PN + RN) + (N + 1) + 2$ is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

6) Effective SQL parameter type list entry $(PN + RN) + (N + 1) + 3$ is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

7) Effective SQL parameter type list entry $(PN + RN) + (N + 1) + 4$ is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

ii) Otherwise, let the *effective SQL parameter type list* be a list of $PN + N + 4$ SQL parameters, as follows:

1) For i ranging from 1 to *PN*, the i-th effective SQL parameter type list entry consists of the i-th <SQL parameter declaration>.

2) Effective SQL parameter list entries $PN + 1$ to $PN + N$ are N occurrences of an SQL parameter of an implementation-defined <data type> that is an exact numeric type with scale 0. The <parameter mode> for the i-th such effective SQL parameter is the same as that of the $i - PN$-th effective SQL parameter.

3) Effective SQL parameter type list entry $(PN + N) + 1$ is an SQL parameter of a <data type> that is character string of length 5 and character set SQL_TEXT with <parameter mode> INOUT.

4) Effective SQL parameter type list entry $(PN + N) + 2$ is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

5) Effective SQL parameter type list entry $(PN + N) + 3$ is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

6) Effective SQL parameter type list entry $(PN + N) + 4$ is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

i) If PARAMETER STYLE GENERAL is specified, then let the effective SQL parameter type list be a list of PN parameters such that for i ranging from 1 to PN, the i-th effective SQL parameter type list entry consists of the i-th <SQL parameter declaration>.

NOTE 26 – The value returned from the external routine is passed to the SQL-implementation in an implementation-dependent manner. An SQL parameter is not used for this purpose.

j) Depending on whether the <language clause> specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences table* be Table 3, "Data type correspondences for Ada", Table 4, "Data type correspondences for C", Table 5, "Data type correspondences for COBOL", Table 6, "Data type correspondences for Fortran", Table 7, "Data type correspondences for MUMPS", Table 8, "Data type correspondences for Pascal", or Table 9, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the "SQL data type" column and the "host data type column".

k) Any <data type> in an <SQL parameter declaration> shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not "none".

25) If DETERMINISTIC is specified, then R is *deterministic*; otherwise, it is *possibly non-deterministic*.

26) An SQL-invoked routine *possibly modifies SQL-data* if and only if <SQL-data access indication> specifies MODIFIES SQL DATA.

27) An SQL-invoked routine *possibly reads SQL-data* if and only if <SQL-data access indication> specifies READS SQL DATA.

28) An SQL-invoked routine *possibly contains SQL* if and only if <SQL-data access indication> specifies CONTAINS SQL.

29) An SQL-invoked routine *does not possibly contain SQL* if and only if <SQL-data access indication> specifies NO SQL.

Access Rules

1) If an <SQL-invoked routine> is contained in a <module> *M* with no intervening <schema definition>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns *S*.

General Rules

1) If *R* is a schema-level routine, then a privilege descriptor is created that defines the EXECUTE privilege on *R* to the <authorization identifier> that owns the the schema that includes *R*. The grantor for the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable if and only if one of the following is satisfied:

   a) *R* is an SQL routine and all of the privileges necessary for the <authorization identifier> to successfully execute the <SQL procedure statement> contained in the <routine body> are grantable. The necessary privileges include the EXECUTE privilege on every subject routine of every <routine invocation> contained in the <SQL procedure statement>.

   b) *R* is an external routine.

2) Case:

   a) If <SQL-invoked routine> is contained in an <SQL-server module definition>, then let *DP* be the SQL-path of that <SQL-server module definition>.

   b) If <SQL-invoked routine> is contained in a <schema definition> without an intervening <SQL-server module definition>, then let *DP* be the SQL-path of that <schema definition>.

   c) If <SQL-invoked routine> is contained in a <preparable statement> or in a <direct SQL statement>, then let *DP* be the SQL-path of the current SQL-session.

   d) Otherwise, let *DP* be the SQL-path of the <module> that contains <SQL-invoked routine>.

3) A routine descriptor is created that describes the SQL-invoked routine being defined:

   a) The routine name included in the routine descriptor is <routine name>.

   b) The specific name included in the routine descriptor is <specific name>.

   c) The routine descriptor includes, for each SQL parameter in <SQL parameter list>, the name, data type, ordinal position, and an indication of whether the SQL parameter is input, output, or both.

   d) The routine descriptor includes an indication of whether the SQL-invoked routine is an SQL-invoked function or an SQL-invoked procedure.

   e) If the SQL-invoked routine is an SQL-invoked function, then the routine descriptor includes the data type in the <returns data type>.

   f) The name of the language in which the body of the SQL-invoked routine was written is the <language name> contained in the <language clause>.

   g) If the SQL-invoked routine is an SQL routine, then the SQL routine body of the routine descriptor is the <SQL routine body>.

   h) If the SQL-invoked routine is an external routine, then the external name of the routine descriptor is <external routine name>.

i) If the SQL-invoked routine is an external routine, then the routine descriptor includes an indication of whether the *parameter passing style* is PARAMETER STYLE SQL or PARAMETER STYLE GENERAL.

j) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine is DETERMINISTIC or NOT DETERMINISTIC.

k) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine's <SQL data access indication> is READS SQL DATA, MODIFIES SQL DATA, CONTAINS SQL, or NO SQL.

l) If the SQL-invoked routine specifies a <result cast>, then the routine descriptor includes an indication that the SQL-invoked routine specifies a <result cast> and the <data type> specified in the <result cast>.

m) If *R* is an external routine, then:

   i) Case:

      1) If <SQL data access indication> is MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL, then:

         A) Let *P* be the program identified by the <external routine name>.

         B) The external routine authorization identifier of *R* is the <module authorization identifier> of the <module> of *P*.

         C) The external routine SQL-path is the <schema name list> immediately contained in the <path specification> that is immediately contained in the <module path specification> of the <module> of *P*.

      2) Otherwise:

         A) The external routine authorization identifier is implementation-defined.

         B) The external routine SQL-path is implementation-defined.

      NOTE 27 – The external routine SQL-path is used to set the routine SQL-path of the current SQL-session when *R* is invoked. The routine SQL-path of the current SQL-session is used by the Syntax Rules of Subclause 9.1, "<routine invocation>", to define the subject routines of <routine invocation>s contained in the <module> of *P*. The same external routine SQL-path is used whenever *R* is invoked.

   ii) The effective SQL parameter list is the *effective SQL parameter list*.

n) The routine authorization identifier is the <authorization identifier> that owns *S*.

o) The routine SQL-path is *DP*.

   NOTE 28 – The routine SQL-path is used to set the routine SQL-path of the current SQL-session when *R* is invoked. The routine SQL-path of the current SQL-session is used by the Syntax Rules of Subclause 9.1, "<routine invocation>", to define the subject routines of <routine invocation>s contained in *R*. The same routine SQL-path is used whenever *R* is invoked.

p) An indication of whether the routine is a schema-level routine.

q) If the SQL-invoked routine is a schema-level routine, then the schema name of the schema that includes the SQL-invoked routine; otherwise, the SQL-server module name of the SQL-server module that includes the SQL-invoked routine and the schema name of the schema that includes that SQL-server module.

10.19 <drop routine statement>

Function

Destroy an SQL-invoked routine.

Format

```
<drop routine statement> ::=
      DROP <specific routine designator> <drop behavior>
```

Syntax Rules

1) Let *SR* be the SQL-invoked routine identified by the <specific routine designator> and let *SN* be the <specific name> of *SR*. The schema identified by the explicit or implicit <schema name> of *SN* shall include the descriptor of *SR*.

2) *SR* shall be a schema-level routine.

3) If RESTRICT is specified, then no <routine invocation> whose subject routines include *SR* shall be contained in any of the following:

   a) The <SQL routine body> of any routine descriptor.

   b) The <query expression> of any view descriptor.

   c) The <search condition> of any constraint descriptor or assertion descriptor.

   d) The module descriptor of any SQL-server module.

   NOTE 29 – If CASCADE is specified, then such referencing objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equivalent to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *SR*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   ```
   REVOKE EXECUTE ON SPECIFIC ROUTINE SN FROM A CASCADE
   ```

2) The identified SQL-invoked routine and its descriptor are destroyed.

10.20 <grant statement>

Function

Define privileges.

Format

```
<object name> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <specific routine designator>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR7) For every <grantee> *G* and for every view *V1* owned by *G*, if *G* has been granted SELECT privilege WITH GRANT OPTION on all tables identified by a <table name> contained in the <query expression> of *V1* and EXECUTE privilege WITH GRANT OPTION on all SQL-invoked routines that are subject routines of a <routine invocation> contained in the <query expression> of *V1*, then for every privilege descriptor with a <privileges> *P* that contains SELECT, a <grantor> of "_SYSTEM", an <object> of *V1*, and <grantee> *G* that is not grantable, the following <grant option> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

```
GRANT P ON V1 TO G WITH GRANT OPTION
```

2) Insert this GR For every <grantee> *G* and for every schema-level SQL-invoked routine *R1* owned by *G*, if the user privileges of *G* contain all of the privileges necessary to successfully execute the <SQL procedure statement> contained in the <routine body> of *R1* WITH GRANT OPTION, then for every privilege descriptor with a <privileges> EXECUTE, a <grantor> of "_SYSTEM", <object> of *R1*, and <grantee> *G* that is not grantable, the following <grant statement> is executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

```
GRANT EXECUTE ON R1 TO G WITH GRANT OPTION.
```

NOTE 30 – The privileges necessary include the EXECUTE privilege on every subject routine of every <routine invocation> contained in the <SQL procedure statement>.

3) Insert this GR For every <grantee> *G* and for every SQL-server module *M1* owned by *G*, if the user privileges of *G* contain all of the privileges necessary to successfully execute every <SQL procedure statement> contained in the <routine body> of every SQL-invoked routine contained in *M1* WITH GRANT OPTION, then for every privilege descriptor with a <privileges> EXECUTE, a <grantor> of "_SYSTEM", <object> of *M1*, and <grantee> *G* that is not grantable,

the following <grant statement> is executed with a current <authorization identifier> of "_ SYSTEM" and without further Access Rule checking:

```
GRANT EXECUTE ON M1 TO G WITH GRANT OPTION.
```

NOTE 31 – The privileges necessary include the EXECUTE privilege on every subject routine of every <routine invocation> contained in those <SQL procedure statement>s.

10.21 <revoke statement>

Function

Destroy privileges.

Format

No additional Format items.

Syntax Rules

1) Augment SR7) An EXECUTE privilege descriptor *D* is *allowed to be created* by a grant permitted by descriptor *P* if the following conditions hold:

   a) *P* indicates that the privilege that it represents is grantable, and

   b) The grantee of *P* is the same as the grantor of *D* or the grantee of *P* is PUBLIC, and

   c) *P* and *D* are both EXECUTE privilege descriptors. The action and the identified SQL-invoked routine of *P* are the same as the action and the identified SQL-invoked routine of *D*.

2) Replace SR12) Let *S1* be the name of any schema and *A1* be the <authorization identifier> that owns the schema identified by *S1*.

3) Replace SR13) Let *V* be any view descriptor included in *S1*. *V* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having:

   a) SELECT privilege on one or more tables contained in the <query expression> of *V*.

   b) USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in the <query expression> of *V*.

   c) EXECUTE privilege on one or more schema-level routines that are among the subject routines of a <routine invocation> that is generally contained in the <query expression> of *V*.

   d) EXECUTE privilege on one or more SQL-server modules that include one or more SQL-invoked routines that are among the subject routines of a <routine invocation> that is generally contained in the <query expression> of *V*.

4) Replace SR14) Let *TC* be any table constraint descriptor included in *S1*. *TC* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having:

   a) REFERENCES privilege on one or more columns referenced in in any <search condition> of *TC*.

   b) USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in any <search condition> of *TC*.

c) EXECUTE privilege on one or more schema-level routines that are among the subject routines of a <routine invocation> that is generally contained in any <search condition> of *TC*.

d) EXECUTE privilege on one or more SQL-server modules that include one or more SQL-invoked routines that are among the subject routines of a <routine invocation> that is generally contained in any <search condition> of *TC*.

5) Replace SR15) Let *AX* be any assertion descriptor included in *S1*. *AX* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having:

a) REFERENCES privilege on one or more columns referenced in any <search condition> of *AX*.

b) USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in any <search condition> of *AX*.

c) EXECUTE privilege on one or more schema-level routines that are among the subject routines of a <routine invocation> that is generally contained in any <search condition> of *AX*.

d) EXECUTE privilege on one or more SQL-server modules that include one or more SQL-invoked routines that are among the subject routines of a <routine invocation> that is generally contained in any <search condition> of *AX*.

6) Replace SR16) Let *DC* be any domain constraint descriptor included in *S1*. *DC* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having:

a) REFERENCES privilege on one or more columns referenced in any <search condition> of *DC*.

b) USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in any <search condition> of *DC*.

c) EXECUTE privilege on one or more schema-level routines that are among the subject routines of a <routine invocation> that is generally contained in any <search condition> of *DC*.

d) EXECUTE privilege on one or more SQL-server modules that include one or more SQL-invoked routines that are among the subject routines of a <routine invocation> that is generally contained in any <search condition> of *DC*.

7) Insert this SR Let *RD* be any routine descriptor included in *S1*. *RD* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer satisfying one or more of the following criteria:

a) Having EXECUTE privilege on one or more SQL-invoked routines that are among the subject routines of a <routine invocation> that is contained in the <routine body> of *RD*.

b) Having EXECUTE privilege on one or more SQL-server modules that include one or more SQL-invoked routines that are among the subject routines of a <routine invocation> that is contained in the <routine body> of RD.

c) Having SELECT privilege on each <table reference> contained in a <query expression> simply contained in a <cursor specification> or an <insert statement> contained in the <SQL routine body> of *RD*.

d) Having SELECT privilege on each <table reference> contained in a <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <SQL routine body> of *RD*.

e) Having SELECT privilege on each <table reference> and <column reference> contained in a <search condition> contained in a <delete statement: searched> or an <update statement: searched> contained in the <SQL routine body> of *RD*.

f) Having SELECT privilege on each <table reference> and <column reference> contained in a <value expression> immediately contained in an <update source> contained in the <SQL routine body> of *RD*.

g) Having INSERT privileges on each column

Case:

i) Named in the <insert column list> of an <insert statement> contained in the <SQL routine body> of *RD*.

ii) Of the table identified by the <table name> immediately contained in an <insert statement> that does not contain an <insert column list> and that is contained in the <SQL routine body> of *RD*.

h) Having UPDATE privileges on each column whose name is contained in an <object column> contained in either an <update statement: positioned> or an <update statement: searched> contained in the <SQL routine body> of *RD*.

i) Having DELETE privileges on each table whose name is contained in a <table name> contained in either a <delete statement: positioned> or a <delete statement: searched> contained in the <SQL routine body> of *RD*.

j) Having USAGE privilege on each domain, collation, character set, and translation whose name is contained in the <SQL routine body> of *RD*.

8) Insert this SR Let *SSM* be any SQL-server module descriptor of an SQL-server module included in *S1*. *SSM* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer satisfying one or more of the following criteria:

a) Having EXECUTE privilege on one or more schema-level routines that are among the subject routines of a <routine invocation> that is generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

b) Having EXECUTE privilege on one or more SQL-server modules that include one or more SQL- invoked routines that are among the subject routines of a <routine invocation> that is generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

c) Having SELECT privileges on each <table reference> contained in a <query expression> simply contained in a <cursor specification> or an <insert statement> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

d) Having SELECT privileges on each <table reference> contained in a <table expression> or <select list> immediately contained in a <select statement: single row> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

e) Having SELECT privileges on each <table reference> and < column reference> contained in a <search condition> contained in a <delete statement: positioned> or an <update statement: searched> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

f) Having SELECT privileges on each <table reference> and <column reference> contained in a <value expression> immediately contained in an <update source> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

g) Having INSERT privileges on each column

Case:

i) Named in the <insert column list> of an <insert statement> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

ii) Of the table identified by the <table name> immediately contained in an <insert statement> that does not contain an <insert column list> and that is generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

h) Having UPDATE privileges on each column whose name is contained in an <object column> contained in either an <update statement: positioned> or an <update statement: searched> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

i) Having DELETE privileges on each table whose name is contained in a <table name> immediately contained in either a <delete statement: positioned> or a <delete statement: searched> generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

j) Having USAGE privileges on each domain, collation, character set, and translation whose name is generally contained in the <routine body> of any SQL-invoked routine included in *SSM*.

9) Augment SR18) If RESTRICT is specified, then there shall be no abandoned routine descriptors.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR8) For every abandoned routine descriptor *RD*, let *R* be the schema-level routine whose descriptor is *RD*. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

2) Insert before GR8) For every abandoned SQL-server module descriptor *MD*, let *M* be the SQL-server module whose descriptor is *MD*. Let *MN* be the <SQL-server module name> of *M*. The

following <drop module statement> is effectively executed without further Access Rule checking:

```
DROP MODULE MN CASCADE
```

11 Modules

11.1 <module>

Function

Define a module.

Format

```
<module> ::=
       <module name clause>
       <language clause>
       <module authorization clause>
       [ <module path specification> ]
       [ <temporary table declaration>... ]
       <module contents>...

<module path specification> ::=
       <path specification>
```

Syntax Rules

1) Insert before SR4) If <module path specification> is not specified, then a <module path specification> containing an implementation-defined <schema name list> that includes the <schema name> contained in <module authorization clause> is implicit.

2) Insert before SR4) The explicit or implicit <catalog name> of each <schema name> contained in the <schema name list> of the <module path specification> shall be equivalent to the <catalog name> of the explicit or implicit <schema name> contained in <module authorization clause>.

3) Insert before SR4) The <schema name list> of the explicit or implicit <module path specification> is used as the SQL-path of the <module>. The SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <module>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

11.2 <procedure>

Function

Define a procedure.

Format

No additional Format items.

Syntax Rules

1) Insert before SR6)b)i) If the <parameter name> *PN* of a parameter *P* is contained in an <SQL argument> A_i of the <SQL argument list> of a <routine invocation> immediately contained in a <call statement> that is contained in <SQL procedure statement>, then:

   a) Let *R* be the subject routine of the <routine invocation>.

   b) Let PR_i be the *i*-th SQL parameter of *R*.

   c) Case:

      i) If *PN* is simply contained in a <parameter specification> that is the <target specification> that is simply contained in A_i and PR_i is an output SQL parameter, then *P* is an output parameter.

      ii) If *PN* is simply contained in a <parameter specification> that is the <target specification> that is simply contained in A_i and PR_i is both an input SQL parameter and an output SQL parameter, then *P* is both an input parameter and an output parameter.

      iii) Otherwise, *P* is an input parameter.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 6) through GR 12) The General Rules of Subclause 12.5, "<SQL procedure statement>" of ISO/IEC 9075:1992 are evaluated with *S* as the executing statement.

11.3 Calls to a <procedure>

Function

Define the call to a <procedure> by an SQL-agent.

Syntax Rules

1) Insert into SR1)c) within **package** SQLSTATE_CODES

```
CASE_NOT_FOUND_FOR_CASE_STATEMENT_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="20000";
DATA_EXCEPTION_NULL_VALUE_NOT_ALLOWED:
        constant SQLSTATE_TYPE :="22004";
EXTERNAL_ROUTINE_EXCEPTION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="38000";
EXTERNAL_ROUTINE_EXCEPTION_CONTAINING_SQL_NOT_PERMITTED:
        constant SQLSTATE_TYPE :="38001";
EXTERNAL_ROUTINE_EXCEPTION_MODIFYING_SQL_DATA_NOT_PERMITTED:
        constant SQLSTATE_TYPE :="38002";
EXTERNAL_ROUTINE_EXCEPTION_PROHIBITED_SQL_STATEMENT_ATTEMPTED:
        constant SQLSTATE_TYPE :="38003";
EXTERNAL_ROUTINE_EXCEPTION_READING_SQL_DATA_NOT_PERMITTED:
        constant SQLSTATE_TYPE :="38004";
EXTERNAL_ROUTINE_INVOCATION_EXCEPTION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="39000";
EXTERNAL_ROUTINE_INVOCATION_EXCEPTION_INVALID_SQLSTATE_RETURNED:
        constant SQLSTATE_TYPE :="39001";
EXTERNAL_ROUTINE_INVOCATION_EXCEPTION_NULL_VALUE_NOT_ALLOWED:
        constant SQLSTATE_TYPE :="39002";
INVALID_SCHEMA_NAME_LIST_SPECIFICATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0E000";
INVALID_TRANSACTION_INITIATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0B000";
RESIGNAL_WHEN_HANDLER_NOT_ACTIVE_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0K000";
SQL_ROUTINE_EXCEPTION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2F000";
SQL_ROUTINE_EXCEPTION_FUNCTION_EXECUTED_NO_RETURN_STATEMENT:
        constant SQLSTATE_TYPE :="2F005";
SQL_ROUTINE_EXCEPTION_MODIFYING_SQL_DATA_NOT_PERMITTED:
        constant SQLSTATE_TYPE :="2F002";
SQL_ROUTINE_EXCEPTION_PROHIBITED_SQL_STATEMENT_ATTEMPTED:
        constant SQLSTATE_TYPE :="2F003";
SQL_ROUTINE_EXCEPTION_READING_SQL_DATA_NOT_PERMITTED:
        constant SQLSTATE_TYPE :="2F004";
UNHANDLED_USER_DEFINED_EXCEPTION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="45000";
```


Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

11.4 <SQL procedure statement>

Function

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

Format

```
<SQL procedure statement> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <SQL control statement>

<SQL schema definition statement> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <SQL-server module definition>
      | <SQL-invoked routine>

<SQL schema manipulation statement> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <drop module statement>
      | <drop routine statement>

<SQL session statement> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <set path statement>

<SQL control statement> ::=
        <call statement>
      | <return statement>
      | <assignment statement>
      | <compound statement>
      | <case statement>
      | <if statement>
      | <leave statement>
      | <loop statement>
      | <while statement>
      | <repeat statement>
      | <for statement>

<SQL diagnostics statement> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | <signal statement>
      | <resignal statement>
```


Syntax Rules

1) Insert this SR An <SQL connection statement> shall not be generally contained in an <SQL control statement>, an <SQL-invoked routine>, or an <SQL-server module definition>.

2) The SQL-invoked routine specified by <SQL-invoked routine> shall be a schema-level routine.
NOTE 32 – "Schema-level routine" is defined in Subclause 10.18, "<SQL-invoked routine>".

3) Insert this SR An <SQL procedure statement> *S* is *possibly non-deterministic* if and only if:

   a) *S* is a <select statement: single row> that is possibly non-deterministic.

b) *S* simply contains a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

c) *S* generally contains a <query specification> or a <query expression> that is possibly non-deterministic.

d) *S* generally contains a <datetime value function>, CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH.

e) *S* is a <compound statement> and *S* contains an <SQL variable declaration> that specifies a <default option> that contains a <datetime value function>, CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH.

4) Insert this SR An <SQL procedure statement> *possibly contains SQL* if and only if at least one of the following is satisfied:

a) It is an SQL-schema statement, and SQL-session statement, an SQL diagnostics statement, or an SQL-control statement.

b) It simply contains a <routine invocation> whose subject routines include an SQL-invoked routine that possibly contains SQL.

5) Insert this SR An <SQL procedure statement> *possibly reads SQL-data* if and only if at least one of the following is satisfied:

a) It is an SQL-data statement.

b) It simply contains a <subquery>.

c) It simply contains a <routine invocation> whose subject routines include an SQL-invoked routine that possibly reads SQL-data.

d) It simply contains an <SQL procedure statement> that possibly reads SQL-data.

6) Insert this SR An <SQL procedure statement> *S possibly modifies SQL-data* if and only if at least one of the following is satisfied:

a) *S* simply contains a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

b) *S* is an <SQL data change statement>.

c) *S* simply contains an <SQL procedure statement> that possibly modifies SQL-data.

7) Insert this SR An <SQL schema statement> shall not be contained in an <SQL-invoked routine> or in an <SQL-server module definition>.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR An *atomic execution context* is active during execution of an <SQL procedure statement> that is not an <SQL control statement>.

2) Insert this GR Let S be the executing statement specified in an application of this Subclause.

3) Insert this GR Case:

   a) If S is immediately contained in a <procedure>, then:

      i) If S is an <SQL connection statement>, then:

         1) The <module> that contains S is associated with the SQL-agent.

         2) The diagnostics area is emptied.

         3) The values of all input parameters to the <procedure> are established as follows:

            A) When a <procedure> is called by an SQL-agent, let PD_i be the <parameter declaration> of the i-th parameter and let DT_i and PN_i be the <data type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the i-th parameter in the procedure call.

            B) The General Rules of Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992, are evaluated for input parameters.

         4) The General Rules of S are evaluated.

         5) The General Rules of Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992, are evaluated for output parameters.

         6) If S successfully initiated or resumed an SQL-session, then subsequent calls to a <procedure> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

      ii) If S is an <SQL diagnostics statement>, then:

         1) The <module> that contains S is associated with the SQL-agent.

         2) The values of all input parameters to the <procedure> are established as follows:

            A) When a <procedure> is called by an SQL-agent, let PD_i be the <parameter declaration> of the i-th parameter and let DT_i and PN_i be the <data type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the i-th parameter in the procedure call.

            B) The General Rules of Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992, are evaluated for input parameters.

         3) The General Rules of S are evaluated.

         4) The General Rules of Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992, are evaluated for output parameters.

iii) Otherwise:

1) If no SQL-session is current for the SQL-agent, then

Case:

A) 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
```

B) 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
```

C) Otherwise, an exception condition is raised: *connection exception — connection does not exist*.

D) Subsequent calls to a <procedure> or 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.

2) If an SQL-transaction is active for the SQL-agent, then *S* is associated with that SQL-transaction.

3) If no SQL-transaction is active for the SQL-agent and *S* is a transaction-initiating SQL-statement, then

A) An SQL-transaction is effectively initiated and associated with this call and with subsequent calls of any <procedure> or invocations of <direct SQL statement>s by that SQL-agent until the SQL-agent terminates that SQL-transaction.

B) Case:

I) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session, then the access mode, constraint mode, and isolation level of the SQL-transaction are set as specified by the <set transaction statement>.

II) Otherwise, the access mode of that SQL-transaction is read-write, the constraint mode for all constraints in that SQL-transaction is immediate, and the isolation level of that SQL-transaction is SERIALIZABLE.

C) The SQL-transaction is associated with the SQL-session.

D) The <module> that contains *S* is associated with the SQL-transaction.

4) The <module> that contains *S* is associated with the SQL-agent.

5) 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*.

6) The diagnostics area is emptied.

7) The values of all input parameters to the <procedure> are established as follows:

A) When a <procedure> is called by an SQL-agent, let PD_i be the <parameter declaration> of the i-th parameter and let DT_i and PN_i be the <data type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the i-th parameter in the procedure call.

B) The General Rules of Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992, are evaluated for input parameters.

8) The General Rules of S are evaluated.

9) If the non-dynamic or dynamic execution of an <SQL data statement> or the execution of an <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*.

10) The General Rules of Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992, are evaluated for output parameters.

11) If S is a <select statement: single row> or a <fetch statement> and a completion condition no data is raised or an exception condition is raised, then the value of each PI_i for which PN_i is referenced in a <target specification> in S is implementation-dependent.

b) Otherwise:

i) If an SQL-transaction is active for the SQL-agent, then S is associated with that SQL-transaction.

ii) If no SQL-transaction is active for the SQL-agent and S is a transaction-initiating SQL-statement, then

1) An SQL-transaction is effectively initiated as follows.

Case:

A) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session, then the access mode, constraint mode, and isolation level of the SQL-transaction are set as specified by the <set transaction statement>.

B) Otherwise, the access mode of that SQL-transaction is read-write, the constraint mode for all constraints in that SQL-transaction is immediate, and the isolation level of that SQL-transaction is SERIALIZABLE.

2) The SQL-transaction is associated with the SQL-session.

iii) If S is 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*.

iv) If S is not an <SQL diagnostic statement>, then the diagnostics area is emptied.

4) Insert this GR Case:

a) If S is immediately contained in a <procedure>, then

Case:

i) If S executed successfully, then

1) If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

2) Either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*, as determined by the General Rules in this and other Subclauses of ISO/IEC 9075:1992.

ii) If S did not execute successfully, then:

1) The status parameter(s) is (are) set to the value(s) specified for the condition in clause Clause 22, "Status codes", in ISO/IEC 9075:1992. If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

2) If S is not an <SQL control statement> or if S is a <compound statement> that specifies ATOMIC, then all changes made to SQL-data or schemas by the execution of S are canceled.

b) Otherwise, the General Rules for S are evaluated.

i) If the non-dynamic or dynamic execution of an <SQL data statement> or the execution of an <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*.

ii) Case:

1) 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*, as determined by the General Rules in this and other Subclauses of this International Standard.

2) Otherwise:

A) If S is not an <SQL control statement> or if S is a <compound statement> that specifies ATOMIC, then all changes made to SQL-data or schemas by the execution of S are canceled.

B) The same exception condition is re-raised as determined by the General Rules in this and other Subclauses of this International Standard.

5) Insert this GR Case:

a) If S is not an <SQL diagnostics statement>, then diagnostics information resulting from the execution of S is placed into the diagnostics area as specified in Clause 18, "Diagnostics management", of ISO/IEC 9075:1992.

b) If *S* is an <SQL diagnostics statement>, then the diagnostics area is not updated.

11.5 Data type correspondences

Function

Specify the data type correspondences for SQL data types and host language types.

NOTE 33 – These tables are referenced in Subclause 10.18, "<SQL-invoked routine>", for the definitions of external routines.

Tables

In the following tables, let *P* be <precision>, *S* be <scale>, *L* be <length>, *T* be <time fractional seconds precision>, and *Q* be <interval qualifier>.

Table 3—Data type correspondences for Ada

SQL Data Type	Ada Data Type
SQLSTATE	SQL_STANDARD.SQLSTATE_TYPE
SQLCODE	SQL_STANDARD.SQLCODE_TYPE
CHARACTER (*L*)	SQL_STANDARD.CHAR, with P'LENGTH of *L*
CHARACTER VARYING (*L*)	*None*
BIT (*L*)	SQL_STANDARD.BIT, with P'LENGTH of *L*
BIT VARYING (*L*)	*None*
SMALLINT	SQL_STANDARD.SMALLINT
INTEGER	SQL_STANDARD.INT
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	SQL_STANDARD.REAL
DOUBLE PRECISION	SQL_STANDARD.DOUBLE_PRECISION
FLOAT(*P*)	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

Table 4—Data type correspondences for C

SQL Data Type	C Data Type
SQLSTATE	char, with length 6
SQLCODE	pointer to long
CHARACTER (*L*)	char, with length (*L*+1)*k[1]
CHARACTER VARYING (*L*)	char, with length (*L*+1)*k[1]
BIT (*L*)	char, with length X[2]
BIT VARYING (*L*)	*None*
SMALLINT	pointer to short
INTEGER	pointer to long
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	pointer to float
DOUBLE PRECISION	pointer to double
FLOAT(*P*)	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

[1] k is the length in units of C **char** of the largest character in the character set associated with the SQL data type.

[2] The length X of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in character of the host language.

Table 5—Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
SQLSTATE	PICTURE X(5)
SQLCODE	PICTURE S9(*PC*) USAGE COMPUTATIONAL, where *PC* is implementation-defined between 4 and 18, inclusive
CHARACTER (*L*)	alphanumeric, with length *L*
CHARACTER VARYING (*L*)	*None*
BIT (*L*)	alphanumeric, with length *X*[1]
BIT VARYING (*L*)	*None*
SMALLINT	PICTURE S9(*SPI*) USAGE BINARY, where *SPI* is implementation-defined
INTEGER	PICTURE S9(*PI*) USAGE BINARY, where *PI* is implementation-defined
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	USAGE DISPLAY SIGN LEADING SEPARATE, with PICTURE as specified[2]
REAL	*None*
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

[1]The length of a character type corresponding with SQL BIT(*L*) is one more than the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in one character of the host language.

[2]Case:

a) If *S*=*P*, then a PICTURE with an 'S' followed by a 'V' followed by *P* '9's.
b) If *P*>*S*>0, then a PICTURE with an 'S' followed by *P*–*S* '9's followed by a 'V' followed by *S* '9's.
c) If *S*=0, then a PICTURE with an 'S' followed by *P* '9's optionally followed by a 'V'.

Table 6—Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
SQLSTATE	CHARACTER, with length 5
SQLCODE	INTEGER
CHARACTER (*L*)	CHARACTER, with length *L*
CHARACTER VARYING (*L*)	*None*
BIT (*L*)	CHARACTER, with length *X*[1]
BIT VARYING (*L*)	*None*
SMALLINT	*None*
INTEGER	INTEGER
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	REAL
DOUBLE PRECISION	DOUBLE PRECISION
FLOAT(*P*)	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in character of the host language.

Table 7—Data type correspondences for MUMPS

SQL Data Type	MUMPS Data Type
SQLSTATE	character, with maximum length at least 5
SQLCODE	*None*
CHARACTER (*L*)	*None*
CHARACTER VARYING (*L*)	character with maximum length *L*
BIT (*L*)	*None*
BIT VARYING (*L*)	*None*
SMALLINT	*None*
INTEGER	character
DECIMAL(*P*,*S*)	character
NUMERIC(*P*,*S*)	character
REAL	character
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

Table 8—Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
SQLSTATE	PACKED ARRAY[1..5] OF CHAR
SQLCODE	INTEGER
CHARACTER(1)	CHAR
CHARACTER (*L*), *L*>1	PACKED ARRAY[1..*L*] OF CHAR
CHARACTER VARYING (*L*)	*None*
BIT (*L*), $1 \leq L \leq B^1$	CHAR
BIT (*L*), $B^1 < L$	PACKED ARRAY[LB[1]] OF CHAR
BIT VARYING (*L*)	*None*
SMALLINT	*None*
INTEGER	INTEGER
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	REAL
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

[1]The length *LB* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a character of the host language.

Table 9—Data type correspondences for PL/I

SQL Data Type	PL/I Data Type
SQLSTATE	CHARACTER(5)
SQLCODE	FIXED BINARY(*PP*), where *PP* is an implementation-defined precision at least 15.
CHARACTER (*L*)	CHARACTER(*L*)
CHARACTER VARYING (*L*)	CHARACTER VARYING(*L*)
BIT (*L*)	BIT(*L*)
BIT VARYING (*L*)	BIT VARYING (*L*)
SMALLINT	FIXED BINARY(*SPI*), where *SPI* is implementation-defined
INTEGER	FIXED BINARY(*PI*), where *PI* is implementation-defined
DECIMAL(*P*,*S*)	FIXED DECIMAL(*P*,*S*)
NUMERIC(*P*,*S*)	*None*
REAL	*None*
DOUBLE PRECISION	*None*
FLOAT(*P*)	FLOAT BINARY (*P*)
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*

12 Data manipulation

12.1 <declare cursor>

Function

Define a cursor.

Format

No additional Format items.

Syntax Rules

1) Replace SR1) If the <declare cursor> is simply contained in the <module contents> of a <module>, then the <cursor name> shall be different from the <cursor name> contained in any other <declare cursor> simply contained in the <module contents> of the same <module>.

NOTE 34 – See the Syntax Rules of Subclause 11.1, "<module>".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.2 <open statement>

Function

Open a cursor.

Format

No additional Format items.

Syntax Rules

1) Replace SR1) Let *CN* be the <cursor name> in the <open statement>. *CN* shall be contained within the scope of one or more <cursor name>s that are equivalent to *CN*. If there is more than one such <cursor name>, then the one with the innermost scope is specified. Let *CR* be the cursor specified by *CN*.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR3)a)ii) Each <datetime value function> generally contained in *S* is replaced by the value resulting from evaluation of that <datetime value function> and each <value specification> generally contained in *S* that is CURRENT_PATH is replaced by the value resulting from evaluation of CURRENT_PATH, with all such evaluations effectively done at the same instant in time.

12.3 <fetch statement>

Function

Position a cursor on a specified row of a table and retrieve values from that row.

Format

No additional Format items.

Syntax Rules

1) Replace SR2) Let *CN* be the <cursor name> in the <fetch statement>. *CN* shall be contained within the scope of one or more <cursor name>s that are equivalent to *CN*. If there is more than one such <cursor name>, then the one with the innermost scope is specified. Let *CR* be the cursor specified by *CN*. Let *T* be the table defined by the <cursor specification> of *CR*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.4 <close statement>

Function

Close a cursor.

Format

No additional Format items.

Syntax Rules

1) Replace SR1) Let *CN* be the <cursor name> in the <close statement>. *CN* shall be contained within the scope of one or more <cursor name>s that are equivalent to *CN*. If there is more than one such <cursor name>, then the one with the innermost scope is specified. Let *CR* be the cursor specified by *CN*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.5 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

No additional Format items.

Syntax Rules

1) Replace SR2) For each <value expression> and corresponding <target specification> as *VALUE* and *TARGET*, respectively,

   Case:

   a) If *TARGET* is an <embedded variable name> or a <parameter specification>, then the Syntax Rules of Subclause 9.1, "Retrieval assignment", of ISO/IEC 9075:1992, shall be satisfied.

   b) If *TARGET* is an <SQL variable name> or the <SQL parameter name> of a parameter of an SQL-invoked routine, then the Syntax Rules of Subclause 9.2, "Store assignment", of ISO/IEC 9075:1992, shall be satisfied.

2) Insert this SR The <table expression> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

3) Insert this SR A <select statement: single row> is *possibly non-deterministic* if it contains a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR8) For each *SV* and corresponding *TV* as *VALUE* and *TARGET*, respectively,

   Case:

   a) If *TARGET* is an <embedded variable name> or a <parameter specification>, then the value of *VALUE* is assigned to *TARGET* according to the General Rules of Subclause 9.1, "Retrieval assignment", of ISO/IEC 9075:1992.

   b) If *TARGET* is an <SQL variable name> or the <SQL parameter name> of a parameter of an SQL-invoked routine, then the value of *VALUE* is assigned to *TARGET* according to the General Rules of Subclause 9.2, "Store assignment", of ISO/IEC 9075:1992.

12.6 <delete statement: positioned>

Function

Delete a row of a table.

Format

No additional Format items.

Syntax Rules

1) Replace SR1) Let *CN* be the <cursor name> in the <delete statement: positioned>. *CN* shall be contained within the scope of one or more <cursor name>s that are equivalent to *CN*. If there is more than one such <cursor name>, then the one with the innermost scope is specified. Let *CR* be the cursor specified by *CN*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.7 <delete statement: searched>

Function

Delete rows of a table.

Format

No additional Format items.

Syntax Rules

1) Insert this SR The <search condition> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.8 <update statement: positioned>

Function

Update a row of a table.

Format

No additional Format items.

Syntax Rules

1) Replace SR1) Let *CN* be the <cursor name> in the <update statement: positioned>. *CN* shall be contained within the scope of one or more <cursor name>s that are equivalent to *CN*. If there is more than one such <cursor name>, then the one with the innermost scope is specified. Let *CR* be the cursor specified by *CN*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.9 <update statement: searched>

Function

Update rows of a table.

Format

No additional Format items.

Syntax Rules

1) Insert this SR The <search condition> shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that possibly modifies SQL-data.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

12.10 <temporary table declaration>

Function

Replace 1st paragraph Declare a declared local temporary table that will be effectively materialized the first time that any <procedure> in the <module> that contains, without an intervening <SQL-server module definition>, the <temporary table declaration> is executed or <module routine> in the <SQL-server module definition> that contains the <temporary table declaration> is executed. The scope of the declared local temporary table is all the <procedure>s of that <module> or <module routine>s of that <SQL-server module definition> executed within the same SQL-session.

Format

No additional Format items.

Syntax Rules

1) Replace SR1) Let *T* be the <local table name> of <qualified local table name>.

Case:

a) If <temporary table declaration> is contained in a <module> without an intervening <SQL-server module definition>, then *T* shall not be equivalent to the <local table name> of any other <temporary table declaration> contained without an intervening <SQL-server module definition> within the <module>.

b) Otherwise, *T* shall not be equivalent to the <local table name> of any other <temporary table declaration> contained within the <SQL-server module definition>.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR1) Case:

a) If <temporary table declaration> is contained in a <module> without an intervening <SQL-server module definition>, then let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and an implementation-dependent name associated with the <module> that contains the <temporary table declaration>.

b) Otherwise, let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and the name of the <SQL-server module definition> that contains the <temporary table declaration>.

2) Replace GR2) Case:

a) If <temporary table declaration> is contained in a <module> without an intervening <SQL-server module definition>, then the definition of *T* within a <module> is effectively equivalent to the definition of a persistent base table *U.T*. Within the module, any reference

to MODULE.*T* that is not contained in an <SQL schema statement> is equivalent to a reference to *U.T*.

b) Otherwise, the definition of *T* within an <SQL-server module definition> is effectively equivalent to the definition of a persistent base table *U.T*. Within the SQL-server module, any reference to MODULE.*T* is equivalent to a reference to *U.T*.

13 Control statements

13.1 <call statement>

Function

Invoke an SQL-invoked routine.

Format

```
<call statement> ::=
       CALL <routine invocation>
```

Syntax Rules

1) Let *RI* be the <routine invocation> immediately contained in the <call statement>.

2) Let *SR* be the set of subject routines specified by applying the Syntax Rules of Subclause 9.1, "<routine invocation>", to *RI*.

3) Every SQL-invoked routine in *SR* shall be an SQL-invoked procedure.

Access Rules

None.

General Rules

1) A subject routine in *SR* is effectively invoked according to the General Rules of Subclause 9.1, "<routine invocation>", with *RI* and *SR* as the <routine invocation> and the set of subject routines, respectively.

13.2 <return statement>

Function

Return a value from an SQL function.

Format

```
<return statement> ::=
       RETURN <return value>

<return value> ::=
         <value expression>
       | <null specification>
```

Syntax Rules

1) <return statement> shall be contained in an <SQL routine body> that is the <routine body> of the <routine specification> of an <SQL-invoked routine> *F* that is an SQL-invoked function. Let *RDT* be the <returns data type> of the <returns clause> of *F*.

2) The <return value> <null specification> is equivalent to the <value expression>:

   CAST (NULL AS *RDT*)

3) Let *VE* be the <value expression> of the <return value> immediately contained in <return statement>.

4) The data type of *VE* shall be assignable to an item of the data type *RDT*, according to the Syntax Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992, with *RDT* and *VE* as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) Let *RV* be the value of *VE*.

2) Let *RT* be an item of data type *RDT*.

3) Let RI_F be a <routine invocation> whose subject routine identifies *F*. The *returned value* of the execution of the <SQL routine body> of *F* is the value resulting from the assignment of *RV* to *RT* according to the General Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992, with *RT* and *RV* as *TARGET* and *VALUE*, respectively.

4) The execution of the <SQL routine body> of *F* is terminated immediately.

13.3 <compound statement>

Function

Specify a statement that groups other statements together.

Format

```
<compound statement> ::=
       [ <beginning label> <colon> ]
       BEGIN [ [ NOT ] ATOMIC ]
       [ <local declaration list> ]
       [ <local cursor declaration list> ]
       [ <local handler declaration list> ]
       [ <SQL statement list> ]
       END [ <ending label> ]

<beginning label> ::= <statement label>

<ending label> ::= <statement label>

<statement label> ::= <identifier>

<local declaration list> ::= <terminated local declaration>...

<terminated local declaration> ::= <local declaration> <semicolon>

<local declaration> ::=
         <SQL variable declaration>
       | <condition declaration>

<local cursor declaration list> ::=
       <terminated local cursor declaration>...

<terminated local cursor declaration> ::=
       <cursor declaration> <semicolon>

<local handler declaration list> ::=
       <terminated local handler declaration>...

<terminated local handler declaration> ::=
       <handler declaration> <semicolon>

<SQL statement list> ::= <terminated SQL statement>...

<terminated SQL statement> ::=
       <SQL procedure statement> <semicolon>
```


Syntax Rules

1) Let *CS* be the <compound statement>.

2) If *CS* is contained in another <compound statement> and *CS* does not specify a <beginning label>, then an implementation-dependent <beginning label> is implicit.

3) If an <ending label> is specified, then *CS* shall specify a <beginning label> that is equivalent to that <ending label>.

4) The scope of the <beginning label> is *CS* excluding every <SQL schema statement> contained in *CS*. <beginning label> shall not be equivalent to any other <beginning label>s contained in *CS* excluding every <SQL schema statement> contained in *CS*.

5) If *CS* specifies neither ATOMIC nor NOT ATOMIC, then NOT ATOMIC is implicit.

6) If *CS* specifies ATOMIC, then the <SQL statement list> shall not generally contain either a <commit statement> or a <rollback statement>.

7) Let *VN* be an <SQL variable name> contained in a <local declaration list>. The *declared local name* of the variable identified by *VN* is *VN*.

8) Let *CN* be the <condition name> immediately contained in a <condition declaration> contained in a <local declaration list>. The *declared local name* of the <condition declaration> is *CN*.

9) Let *CN* be the <cursor name> immediately contained in a <declare cursor> *DC* contained in a <local cursor declaration list>. The *declared local name* of the cursor declared by *DC* is *CN*.

10) No two variables declared in a <local declaration list> shall have equivalent declared local names.

11) No two <condition declaration>s contained in a <local declaration list> shall have equivalent declared local names.

12) No two cursors declared in a <local cursor declaration list> shall have equivalent declared local names.

13) The scope of an <SQL variable name> of an <SQL variable declaration> simply contained in a <local declaration> simply contained in *CS* is the <local cursor declaration list> of *CS*, the <local handler declaration list> of *CS* excluding every <SQL schema statement> contained in the <local handler declaration list> of *CS*, and the <SQL statement list> of *CS* excluding every <SQL schema statement> contained in the <SQL statement list> of *CS*.

14) The scope of the <condition name> in a <condition declaration> simply contained in a <local declaration> simply contained in *CS* is the <local handler declaration list> of *CS* excluding every <SQL schema statement> contained in the <local handler declaration list> of *CS* and the <SQL statement list> of *CS* excluding every <SQL schema statement> contained in the <SQL statement list> of *CS*.

15) The scope of the <cursor name> in a <declare cursor> simply contained in a <cursor declaration> simply contained in *CS* is the <local handler declaration list> of *CS* excluding every <SQL schema statement> contained in the <local handler declaration list> of *CS* and the <SQL statement list> of *CS* excluding every <SQL schema statement> contained in the <SQL statement list> of *CS*.

16) The scope of a <handler declaration> simply contained in a <local handler declaration list> simply contained in *CS* is the <SQL statement list> of *CS* excluding every <SQL schema statement> contained in the <SQL statement list> of *CS*.

17) If the <compound statement> simply contains a <handler declaration> that specifies UNDO, then ATOMIC shall be specified.

Access Rules

None.

General Rules

1) If *CS* specifies ATOMIC, then an *atomic execution context* is active during the execution of *CS*.

2) The variables, cursors, and handlers specified in the <local declaration list>, <local cursor declaration list>, and the <local handler declaration list> of *CS* are created in an implementation-dependent order.

3) Let *N* be the number of <SQL procedure statement>s contained in the <SQL statement list> that is immediately contained in *CS* without an intervening <SQL control statement>. For *i* ranging from 1 to *N*:

   a) Let S_i be the *i*-th such <SQL procedure statement>.

   b) The General Rules of Subclause 12.5, <SQL procedure statement>, in ISO/IEC 9075:1992, are evaluated with S_i as the *executing statement*.

   c) If the execution of S_i terminates with exception conditions or completion conditions other than *successful completion*, then:

      i) The following <resignal statement> is effectively executed without further Syntax Rule checking:

         ```
         RESIGNAL
         ```

      ii) If there are unhandled exception conditions or completion conditions other than *successful completion* at the completion of the execution of a handler (if any), then the execution of *CS* is terminated immediately.

         1) For every open cursor *CR* that is declared in the <local declaration list> of *CS*, the following SQL-statement is effectively executed:

            ```
            CLOSE CR
            ```

         2) The SQL variables, cursors, and handlers specified in the <local declaration list>, the <local cursor declaration list>, and the <local handler declaration list> of *CS* are destroyed.

4) For every open cursor *CR* that is declared in the <local cursor declaration list> of *CS*, the following statement is effectively executed:

   ```
   CLOSE CR
   ```

5) The variables, cursors, and handlers specified in <local declaration list>, the <local cursor declaration list>, and the <local handler declaration list> of *CS* are destroyed.

6) The <condition name> of every <condition declaration> contained in <local declaration list> ceases to be considered to be defined.

13.4 <handler declaration>

Function

Associate a handler with exception or completion conditions to be handled in a compound statement.

Format

```
<handler declaration> ::=
       DECLARE <handler type> HANDLER
         FOR <condition value list>
         <handler action>

<handler type> ::=
         CONTINUE
       | EXIT
       | UNDO

<handler action> ::=
         <SQL procedure statement>

<condition value list> ::=
       <condition value> [ { <comma> <condition value> }... ]

<condition value> ::=
         <sqlstate value>
       | <condition name>
       | SQLEXCEPTION
       | SQLWARNING
       | NOT FOUND
```

Syntax Rules

1) Let *HD* be the <handler declaration>.

2) A <condition name> *CN* specified in a <condition value> of a <handler declaration> shall be defined by some <condition declaration> with a scope that contains *HD*. Let *C* be the condition specified by the innermost such <condition declaration>.

3) If a <condition value> specifies SQLEXCEPTION, SQLWARNING, or NOT FOUND, then neither <sqlstate value> nor <condition value> shall be specified.

4) No other <handler declaration> with the same scope as *HD* shall contain in its <condition value list> a <condition value> that represents the same condition as a <condition value> contained in the <condition value list> of *HD*.

5) The <condition value list> shall not contain the same <condition value> or <sqlstate value> more than once, nor shall it contain both the <condition name> of a condition *C* and an <sqlstate value> that represents the SQLSTATE value associated with *C*.

6) SQLEXCEPTION corresponds to SQLSTATE values with a class value other than "00", "01", and "02" in Subclause 22.1, "SQLSTATE", in ISO/IEC 9075:1992. NOT FOUND and SQLWARNING correspond to SQLSTATE values with class values of "02" and "01", respectively, in Subclause 22.1, "SQLSTATE", in ISO/IEC 9075:1992.

7) If a <condition value> specifies SQLEXCEPTION, SQLWARNING, or NOT FOUND, then the <handler declaration> is a *general* <*handler declaration*>; otherwise, the <handler declaration> is a *specific* <*handler declaration*>.

8) If there is a general <handler declaration> and a specific <handler declaration> for the same <condition value> in the same scope, then only the specific <handler declaration> is associated with that <condition value>.

9) Let *HA* be the <handler action>.

10) *HA* is associated with every <condition name> specified in the <condition value list> of *HD* and with every SQLSTATE value specified in every <sqlstate value> specified in the <condition value list> of *HD*.

11) If *HA* is associated with a <condition name> and that <condition name> was defined for an SQLSTATE value, then *HA* is also associated with that SQLSTATE value.

12) If *HA* is associated with SQLEXCEPTION, then it is associated with each SQLSTATE value having a class value other than "00", "01", or "02". If *HA* is associated with NOT FOUND or SQLWARNING, then it is associated with each SQLSTATE value that has a class value of "02" or "01", respectively.

Access Rules

None.

General Rules

1) When the handler *H* associated with the conditions specified by *HD* is created, it is the *most appropriate handler* for any condition *CN* raised during execution of any SQL-statements that are in the scope of *HD* that has an SQLSTATE value or condition name that is the same as an SQLSTATE value or condition name associated with this handler, until *H* is destroyed. *CN* has a more appropriate handler if during the existence of *H*, another handler *AH* is created with a scope containing *CN*, and if *AH* is associated with an SQLSTATE value or condition name that is the same as the SQLSTATE value or condition name of *CN*. *AH* replaces *H* as the most appropriate handler for *CN* until *AH* is destroyed. When *AH* is destroyed, *H* is reinstated as the most appropriate handler for *CN*.

2) Let *CS* be the <compound statement> simply containing *HD*. Let *CC* be the <compound statement> from which *H* was activated.

3) When *H* is activated,

   Case:

   a) If *HD* specifies CONTINUE, then:

      i) *HA* is executed.

      ii) If there is an unhandled condition other than *successful completion* at the completion of *HA*, then the following <resignal statement> is effectively executed:

```
RESIGNAL
```

      Otherwise, *HA* completes with completion condition *successful completion* and control is returned to the SQL-statement following the one that raised the condition in *CC*.

b) If *HD* specifies EXIT, then:

i) *HA* is executed.

ii) If there is an unhandled condition other than *successful completion* at the completion of *HA*, then the following <resignal statement> is effectively executed:

```
RESIGNAL
```

Otherwise, *HA* completes with completion condition *successful completion* and control is returned to the end of *CS*.

c) If *HD* specifies UNDO, then:

i) All changes made to SQL-data or schemas by the execution of SQL-statements contained in the <SQL statement list> of *CS* and any <SQL procedure statement>s triggered by the execution of any such statements are canceled.

ii) *HA* is executed.

iii) If there is an unhandled condition other than *successful completion* at the completion of *HA*, then the following <resignal statement> is effectively executed:

```
RESIGNAL
```

Otherwise, *HA* completes with completion condition *successful completion* and control is returned to the end of *CS*.

13.5 <condition declaration>

Function

Declare a condition name and an optional corresponding SQLSTATE value.

Format

```
<condition declaration> ::=
       DECLARE <condition name> CONDITION
         [ FOR <sqlstate value> ]
```

Syntax Rules

1) Let *CD* be the <condition declaration>.

2) No other <condition declaration> with the same scope as *CD* shall contain the same <sqlstate value> as *CD*.

Access Rules

None.

General Rules

1) <condition name> is *considered to be defined* for the SQLSTATE value specified by <sqlstate value>.

13.6 <SQL variable declaration>

Function

Declare one or more variables.

Format

```
<SQL variable declaration> ::=
      DECLARE <SQL variable name list>
        <data type> [ <default clause> ]

<SQL variable name list> ::=
      <SQL variable name> [ { <comma> <SQL variable name> }... ]
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) When the variable associated with the <SQL variable declaration> is created, its default value *DV* is derived according to the General Rules of Subclause 10.3, "<default clause>". Let *SV* be the variable defined by the <SQL variable declaration>. The value of *SV* is set to *DV* by the effective invocation of the following SQL-statement:

   ```
   SET SV = DV
   ```

13.7 <assignment statement>

Function

Assign a value to an SQL variable, SQL parameter, host parameter, or host variable.

Format

```
<assignment statement> ::=
       SET <assignment target> <equals operator> <assignment source>

<assignment target> ::=
         <target specification>

<assignment source> ::=
         <value expression>
       | <null specification>
```

Syntax Rules

1) If the <assignment target> simply contains an <embedded variable name> or a <parameter specification>, then <assignment source> shall not simply contain an <embedded variable name> or a <parameter specification>.

2) If the <assignment target> simply contains an <SQL variable name> or the <SQL parameter name> of an SQL parameter of an SQL-invoked routine and the <assignment source> is a <value expression>, then the Syntax Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992 are applied to <assignment target> and <assignment source> as *TARGET* and *VALUE*, respectively.

3) If the <assignment target> simply contains an <embedded variable name> or a <parameter specification> and the <assignment source> is a <value expression>, then the Syntax Rules of Subclause 9.1, "Retrieval assignment", in ISO/IEC 9075:1992 are applied to <assignment target> and <assignment source> as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) If <assignment target> simply contains the <SQL variable name> of an SQL variable *T* or the <SQL parameter name> of an SQL parameter *T* of an SQL-invoked routine, then the value of <assignment source> is assigned to *T* according to the General Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075:1992, with <assignment source> and *T* as *VALUE* and *TARGET*, respectively.

2) If <assignment target> simply contains the <embedded variable name> of a host variable *T* or the <parameter specification> of a parameter *T*, then the value of <assignment source> is assigned to *T* according to the General Rules of Subclause 9.1, "Retrieval assignment", in ISO/IEC 9075:1992, with <assignment source> and *T* as *VALUE* and *TARGET*, respectively.

13.8 <case statement>

Function

Provide conditional execution based on truth of <search condition>s or on equality of operands.

Format

```
<case statement> ::=
        <simple case statement>
      | <searched case statement>

<simple case statement> ::=
      CASE <simple case operand 1>
        <simple case statement when clause>...
        [ <case statement else clause> ]
      END CASE

<searched case statement> ::=
      CASE
        <searched case statement when clause>...
        [ <case statement else clause> ]
      END CASE

<simple case statement when clause> ::=
      WHEN <simple case operand 2>
        THEN <SQL statement list>

<searched case statement when clause> ::=
      WHEN <search condition>
        THEN <SQL statement list>

<case statement else clause> ::=
      ELSE <SQL statement list>

<simple case operand 1> ::= <value expression>

<simple case operand 2> ::= <value expression>
```

Syntax Rules

1) If a <case statement> specifies a <simple case statement>, then let *SCO1* be the <simple case operand 1>:

   a) *SCO1* shall not generally contain a <routine invocation> whose subject routines include an SQL-invoked routine that is possibly non-deterministic or that possibly modifies SQL-data.

   b) The data type of each <simple case operand 2> *SCO2* shall be comparable with the data type of *SCO1*.

   c) The <simple case statement> is equivalent to a <searched case statement> in which each <searched statement when clause> specifies a <search condition> of the form:

      SCO1 = *SCO2*

Access Rules

None.

General Rules

1) Case:

   a) If the <search condition> of some <searched case statement when clause> in a <case statement> is true, then let *SL* be the <SQL statement list> of the first (leftmost) <searched case statement when clause> whose <search condition> is true.

   b) If the <case statement> simply contains a <case statement else clause>, then let *SL* be the <SQL statement list> of that <case statement else clause>.

   c) Otherwise, an exception condition is raised: *case not found for case statement*, and the execution of the <case statement> is terminated immediately.

2) Let *N* be the number of <SQL procedure statement>s simply contained in *SL* without an intervening <SQL control statement>. For *i* ranging from 1 to *N*:

   a) Let S_i be the *i*-th such <SQL procedure statement>.

   b) The General Rules of Subclause 12.5, <SQL procedure statement>, in ISO/IEC 9075:1992, are evaluated with S_i as the *executing statement*.

   c) If the execution of S_i terminates with an unhandled exception condition, then the execution of the <case statement> is terminates with that condition.

13.9 <if statement>

Function

Provide conditional execution based on the truth value of a condition.

Format

```
<if statement> ::=
       IF <search condition>
         <if statement then clause>
         [ <if statement elseif clause>... ]
         [ <if statement else clause> ]
       END IF

<if statement then clause> ::=
       THEN <SQL statement list>

<if statement elseif clause> ::=
       ELSEIF <search condition> THEN <SQL statement list>

<if statement else clause> ::=
       ELSE <SQL statement list>
```

Syntax Rules

1) If one or more <if statement elseif clause>s are specified, then the <if statement> is equivalent to an <if statement> that does not contain ELSEIF by performing the following transformation recursively:

```
IF <search condition>
  <if statement then clause>
  <if statement elseif clause 1>
  [ <if statement elseif clause>  . . .  ]
  [ <if statement else clause> ]
END IF
```

is equivalent to

```
IF <search condition>
  <if statement then clause>
  ELSE
    IF <search condition 1>
    THEN <statement list 1>
      [ <if statement elseif clause>... ]
      [ <if statement else clause> ]
    END IF
END IF
```

where <search condition 1> is the <search condition> directly contained in <if statement elseif clause 1> and <statement list 1> is the <SQL statement list> directly contained in <if statement elseif clause 1>.

Access Rules

None.

General Rules

1) Case:

   a) If the <search condition> immediately contained in the <if statement> evaluates to *true*, then let *SL* be the <SQL statement list> immediately contained in the <if statement then clause>.

   b) Otherwise, if an <if statement else clause> is specified, then let *SL* be the <SQL statement list> immediately contained in the <if statement else clause>.

   NOTE 35 – "Otherwise" means that the <search condition> immediately contained in the <if statement> evaluates to *false* or to *unknown*.

2) Let *N* be the number of <SQL procedure statement>s simply contained in *SL* without an intervening <SQL control statement>. For *i* ranging from 1 to *N*:

   a) Let S_i be the *i*-th such <SQL procedure statement>.

   b) The General Rules of Subclause 12.5 <SQL procedure statement>, in ISO/IEC 9075:1992, are evaluated with S_i as the *executing statement*.

   c) If the execution of S_i terminates with an unhandled exception condition, then the execution of the <case statement> is terminated and the condition remains active.

13.10 <leave statement>

Function

Continue execution by leaving labeled statement.

Format

```
<leave statement> ::=
      LEAVE <statement label>
```

Syntax Rules

1) <statement label> shall be the <beginning label> of some <SQL procedure statement> *S* that contains <leave statement> *L* without an intervening <SQL-schema statement>.

Access Rules

None.

General Rules

1) For every <compound statement> *CS* that is contained in *S* and that contains the <leave statement>:

   a) For every open cursor *CR* that is declared in the <local cursor declaration list> of *CS*, the following statement is effectively executed:

      ```
      CLOSE CR
      ```

   b) The variables, cursors, and handlers specified in the <local declaration list>, the <local cursor declaration list>, and the <local handler declaration list> of *CS* are destroyed.

2) The execution of *S* is terminated.

13.11 <loop statement>

Function

Repeat the execution of a statement.

Format

```
<loop statement> ::=
       [ <beginning label> <colon> ]
       LOOP
         <SQL statement list>
       END LOOP [ <ending label> ]
```

Syntax Rules

1) If <ending label> is specified, then a <beginning label> shall be specified that is equivalent to <ending label>.

2) Let *LS* be the <loop statement>. The scope of the <beginning label> is *LS* excluding every <SQL schema statement> contained in *LS*. <beginning label> shall not be equivalent to any other <beginning label> contained in *LS* excluding every <SQL schema statement> contained in *LS*.

Access Rules

None.

General Rules

1) Let *SSL* be the <SQL statement list> and let *CCS* be the <compound statement>

```
BEGIN NOT ATOMIC SSL END
```

the General Rules of Subclause 12.5, <SQL procedure statement>, of ISO/IEC 9075:1992, are evaluated repeatedly with *CCS* as the *executing statement*.

NOTE 36 – The occurrence of an exception condition or the execution of a <leave statement> may also cause execution of *LS* to be terminated; see Subclause 3.3.4.1, "Exceptions", in ISO/IEC 9075:1992, and Subclause 13.10, "<leave statement>", in this part of ISO/IEC 9075, respectively. Some actions taken by a condition handler might also cause execution of *LS* to be terminated; see Subclause 13.4, "<handler declaration>", in this part of ISO/IEC 9075.

13.12 <while statement>

Function

While a specified condition is true, repeat the execution of a statement.

Format

```
<while statement> ::=
       [ <beginning label> <colon> ]
       WHILE <search condition> DO
         <SQL statement list>
       END WHILE [ <ending label> ]
```

Syntax Rules

1) If <ending label> is specified, then a <beginning label> shall be specified that is equivalent to <ending label>.

2) Let *WS* be the <while statement>. The scope of the <beginning label> is *WS* excluding every <SQL schema statement> contained in *WS*. <beginning label> shall not be equivalent to any other <beginning label> contained in *WS* excluding every <SQL schema statement> contained in *WS*.

Access Rules

None.

General Rules

1) The <search condition> is evaluated.

2) Case:

   a) If the <search condition> evaluates to *false* or *unknown*, then execution of *WS* is terminated.

   b) Let *SSL* be the <SQL statement list> and let *CCS* be the <compound statement>

      BEGIN NOT ATOMIC *SSL* END

      If the <search condition> evaluates to *true*, then the General Rules of Subclause 12.5, <SQL procedure statement>, of ISO/IEC 9075:1992, are evaluated with *CCS* as the *executing statement* and the execution of *WS* is repeated.

      NOTE 37 – The occurrence of an exception condition or the execution of a <leave statement> may also cause execution of *WS* to be terminated; see Subclause 3.3.4.1, "Exceptions", in ISO/IEC 9075:1992, and Subclause 13.10, "<leave statement>", in this part of ISO/IEC 9075, respectively. Some actions taken by a condition handler might also cause execution of *LS* to be terminated; see Subclause 13.4, "<handler declaration>", in this part of ISO/IEC 9075.

13.13 <repeat statement>

Function

Repeat the execution of a statement.

Format

```
<repeat statement> ::=
       [ <beginning label> <colon> ]
       REPEAT
         <SQL statement list>
         UNTIL <search condition>
       END REPEAT [ <ending label> ]
```

Syntax Rules

1) If <ending label> is specified, then a <beginning label> shall be specified that is equivalent to <ending label>.

2) Let *RS* be the <repeat statement>. The scope of the <beginning label> is *RS* excluding every <SQL schema statement> contained in *RS*. <beginning label> shall not be equivalent to any other <beginning label> contained in *RS* excluding every <SQL schema statement> contained in *RS*.

Access Rules

None.

General Rules

1) Let *SSL* be the <SQL statement list> and let *CCS* be the <compound statement>

   BEGIN NOT ATOMIC *SSL* END

   the General Rules of Subclause 12.5, <SQL procedure statement>, of ISO/IEC 9075:1992, are evaluated with *CCS* as the *executing statement* and then <search condition> is evaluated.

   NOTE 38 – The occurrence of an exception condition or the execution of a <leave statement> may also cause execution of *RS* to be terminated; see Subclause 3.3.4.1, "Exceptions", in ISO/IEC 9075:1992, and Subclause 13.10, "<leave statement>", in this part of ISO/IEC 9075, respectively. Some actions taken by a condition handler might also cause execution of *LS* to be terminated; see Subclause 13.4, "<handler declaration>", in this part of ISO/IEC 9075.

2) If the <search condition> evaluates to *false* or *unknown*, then the execution of *RS* is repeated; otherwise, execution of *RS* is terminated.

13.14 <for statement>

Function

Execute a statement for each row of a table.

Format

```
<for statement> ::=
       [ <beginning label> <colon> ]
         FOR <for loop variable name> AS
         [ <cursor name> [ INSENSITIVE ] CURSOR FOR ]
         <cursor specification>
           DO <SQL statement list>
       END FOR [ <ending label> ]

<for loop variable name> ::= <identifier>
```

Syntax Rules

1) Let *FCS* be the <cursor specification> of the <for statement> *FS*.

2) If <ending label> is specified, then a <beginning label> shall be specified that is equivalent to <ending label>.

3) If <cursor name> is specified, then let *CN* be that <cursor name>. Otherwise, let *CN* be an implementation-dependent <cursor name> that is different from any other <cursor name> in the outermost containing <module> or <SQL-invoked routine>.

4) Let *QE* be the <query expression> of *FCS*. Each column of the table specified by *QE* shall have a <column name> that is different from every other <column name> in the table specified by *QE*. Let *V1*, *V2*, . . . , *VN* be those <column name>s. Let *DT1*, *DT2*, . . . , *DTN* be the data types of the respective columns.

5) Let *BL*, *FLVN*, and *SLL* be the <beginning label>, <for loop variable name>, and <SQL statement list> of *FS*, respectively.

   a) If *BL* is not specified, then let *BL* be an implementation-dependent <statement label> that is different from any other <statement label> contained in the outermost containing <SQL control statement>.

   b) Let *AT_END* be an implementation-dependent <SQL variable name> that is different from any other <SQL variable name> or any <SQL parameter name> contained in the outermost containing <SQL-server module>, <SQL-invoked routine>, or <compound statement>.

   c) Let *NOT_FOUND* be an implementation-dependent <condition name> that is different from any other <condition name> contained in the outermost containing <SQL-invoked routine> or <compound statement>.

   d) Let *CS* be INSENSITIVE if INSENSITIVE was specified; otherwise, let *CS* be the empty string.

The <for statement> is equivalent to:

```
BL: BEGIN

  FLVN: BEGIN
    DECLARE CN CS CURSOR FOR FCS;
    DECLARE V1 DT1;
    DECLARE V2 DT2;
.
.
.
    DECLARE VN DTN;
    DECLARE AT_END CHARACTER(1) DEFAULT 'N';
    DECLARE NOT_FOUND CONDITION FOR SQLSTATE '02000';

    BEGIN
      DECLARE CONTINUE HANDLER FOR NOT_FOUND
          SET AT_END = 'Y';

      OPEN CN;
      FETCH CN INTO V1, V2, . . . , VN;
      WHILE AT_END <> 'Y' DO
        SLL;
        BEGIN
          FETCH CN INTO V1, V2, . . . , VN;
        END;
      END WHILE;
      CLOSE CN;
    END;
  END FLVN;
END BL
```

6) *SLL* shall not contain without an intervening <SQL-invoked routine> a <leave statement> that specifies *FLVN*.

7) *SLL* shall not contain without an intervening <SQL-invoked routine> a <fetch statement>, an <open statement>, or a <close statement> that specifies *CN*.

Access Rules

None.

General Rules

None.

14 Transaction management

14.1 <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 If an atomic execution context is active, then an exception condition is raised: *invalid transaction termination*.

14.2 <rollback statement>

Function

Terminate the current SQL-transaction with rollback.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR If an atomic execution context is active, then an exception condition is raised: *invalid transaction termination*.

15 Session management

15.1 <set path statement>

Function

Set the default SQL-path used to determine the subject routine of <routine invocation>s with unqualified <routine name>s contained 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. The default SQL-path remains the current default SQL-path of the SQL-session until another default SQL-path is successfully set.

Format

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

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

Syntax Rules

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

Access Rules

None.

General Rules

1) Let *S* be the character string that is the value of the <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) Let *SNS* be the set of <schema name>s identified by *V*. If any <schema name> *SN* contained in *SNS* does not identify a schema contained in the default catalog of the current SQL_ session, then an exception condition is raised: *invalid schema name list specification*.

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

   NOTE 39 – A <set path statement> that is executed between a <prepare statement> and an <execute statement> has no effect on the prepared statement.

16 Dynamic SQL

16.1 Description of SQL item descriptor areas

Function

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

Syntax Rules

No additional Syntax Rules.

Table 10—Data types of <key word>s used in SQL item descriptor areas

<key word>	Data Type
All alternatives from ISO/IEC 9075:1992	
PARAMETER_MODE	exact numeric with scale 0
PARAMETER_ORDINAL_POSITION	exact numeric with scale 0
PARAMETER_SPECIFIC_CATALOG	character string with character set SQL_TEXT and length not less than 128 characters
PARAMETER_SPECIFIC_SCHEMA	character string with character set SQL_TEXT and length not less than 128 characters
PARAMETER_SPECIFIC_NAME	character string with character set SQL_TEXT and length not less than 128 characters

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR Table 11, "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 11—Codes used for input/output SQL parameter modes in Dynamic SQL

Parameter mode	Code
PARAMETER_MODE_IN	1

Table 11—Codes used for input/output SQL parameter modes in Dynamic SQL (Cont.)

Parameter mode	Code
PARAMETER_MODE_INOUT	2
PARAMETER_MODE_OUT	4

16.2 <prepare statement>

Function

Prepare a statement for execution.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR5)u) If <value specification> in <set catalog statement>, <set schema statement>, <set names statement>, <set session authorization identifier statement>, or <set path statement> is *E1*, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

16.3 <using clause>

Function

Describe the input/output variables for an <SQL dynamic statement>.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR When a <describe input statement> is executed and the prepared statement is a <call statement>, a descriptor for the <dynamic parameter specification>s for the prepared statement is stored in the specified SQL descriptor areas as follows:

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

   b) Case:

      i) If *SR* contains a single SQL-invoked routine *R*, then:

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

         2) Let P_y be the y-th parameter of *R*.
            NOTE 40 – A *P* whose <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.

         3) The PARAMETER_MODE value in the descriptor for each D_x is set to the value from Table 11, "Codes used for input/output SQL parameter modes in Dynamic SQL", that indicates the <SQL parameter mode> of P_y.

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

         5) 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 *R*.

ii) Otherwise, the values of PARAMETER_MODE, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, and PARAMETER_SPECIFIC_NAME in the descriptor for each <dynamic parameter specification> simply contained in the <call statement> are set to implementation-defined values.

2) Insert this GR When a <using clause> is used as the <parameter using clause> in an <execute statement> whose prepared statement PS is a <call statement>, the <using clause> describes the input <dynamic parameter specification> values for the <call statement>.

Let *D* be the number of <dynamic parameter specification>s in *PS* whose PARAMETER_MODE would be set to PARAMETER_MODE_IN or PARAMETER_MODE_INOUT by a <describe input statement> on *PS*.

NOTE 41 – See the General Rules of Subclause 17.8, "<describe statement>", in ISO/IEC 9075:1992.

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

b) If <using descriptor> is specified, then:

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

ii) If the value of COUNT is not *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

iii) If the first *D* item descriptor areas are not valid as specified in Subclause 17.1, "Description of SQL item descriptor areas", in ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

iv) If the value of INDICATOR is not negative, and the value of DATA is not a valid value of the data type indicated by TYPE, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

c) Let *TDT* be the effective data type of the *i*-th input <dynamic parameter specification>, defined to be the type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would be set by a <describe input statement> to reflect the descriptor of the *i*-th input dynamic parameter of *PS*.

NOTE 42 – See the General Rules of Subclause 17.8, "<describe statement>", in ISO/IEC 9075:1992.

d) Case:

i) If <using descriptor> is specified, then let *SDT* be the effective data type of the *i*-th input <dynamic parameter specification> as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME in the *i*-th item descriptor area. If the INDICATOR value of the *i*-th item descriptor area is not negative, then let *SV* be the value represented by the value of DATA with data type *SDT*. Otherwise, let *SV* be the null value.

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

e) If the <cast specification>

```
CAST ( SV AS TDT )
```

violates the Syntax Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

f) If the <cast specification>

```
CAST (SV AS TDT )
```

violates the General Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992, then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992.

g) The <cast specification>

```
CAST ( SV AS TDT )
```

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

h) The values of NAME and UNNAMED are ignored.

3) Insert this GR When a <using clause> is used as the <result using clause> of an <execute statement> whose prepared statement *PS* is a <call statement>, the <using clause> describes the <target specifications>s for the output <dynamic parameter specification>s for *PS*.

Let *D* be the number of <dynamic parameter specification>s in *PS* whose PARAMETER_MODE would be set to PARAMETER_MODE_INOUT or PARAMETER_MODE_OUT by a <describe input statement> on *PS*.

NOTE 43 – See the General Rules of Subclause 17.8, "<describe statement>", in ISO/IEC 9075:1992.

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

b) If <using descriptor> is specified, then:

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

ii) If COUNT is not equal to *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

iii) If the first *D* item descriptor areas are not valid as specified in Subclause 17.1, "Description of SQL item descriptor areas", in ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

4) Insert this GR When a <using clause> is used as the <result using clause> of an <execute statement> whose prepared statement *PS* is a <call statement>, the result is a set of <target specification> values corresponding to the output <dynamic parameter specification>s in

PS. If <using descriptor> is specified, then the SQL descriptor area is set. The value of the *i*-th <target specification> is represented in the *i*-th item descriptor area as follows:

a) Let *SDT* be the effective data type of the *i*-th output <dynamic parameter specification>, defined to be the type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_PRECISION, DATETIME_INTERVAL_CODE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would be set by a <describe input statement> to reflect the description of the *i*-th output <dynamic parameter specification>. Let *SV* be the value of the <dynamic parameter specification>, with data type *SDT*.

NOTE 44 – Note: See the General Rules of Subclause 17.8, "<describe statement>", in ISO/IEC 9075:1992.

b) Let *TDT* be the effective data type of the *i*-th <target specification> as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME, in the *i*-th item descriptor area.

c) If the <cast specification>

```
CAST ( SV AS TDT )
```

violates the Syntax Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

d) If the <cast specification>

```
CAST ( SV AS TDT )
```

violates the General Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992, then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>", in ISO/IEC 9075:1992.

e) The <cast specification>

```
CAST ( SV AS TDT )
```

is effectively performed, and is the value *TV* of the *i*-th <target specification>.

f) If *TV* is the null value, then the value of INDICATOR is set to -1.

g) If *TV* is not the null value, then:

i) The value of INDICATOR is set to 0 and the value of DATA is set to *TV*.

ii) Case:

1) If TYPE indicates CHARACTER VARYING or BIT VARYING, then RETURNED_LENGTH is set to the length in characters or bits, respectively, of *TV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *TV*.

2) If *SDT* is CHARACTER VARYING or BIT VARYING, then RETURNED_LENGTH is set to the length in characters or bits, respectively, of *SV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *SV*.

NOTE 45 – All other values of the SQL descriptor area are unchanged.

17 Diagnostics management

17.1 <get diagnostics statement>

Function

Get exception or completion condition information from the diagnostics area.

Format

```
<condition information item name> ::=
        !! All alternatives from ISO/IEC 9075:1992
      | CONDITION_IDENTIFIER
      | ROUTINE_CATALOG
      | ROUTINE_SCHEMA
      | ROUTINE_NAME
      | SPECIFIC_NAME
      | PARAMETER_NAME
```

Syntax Rules

Table 12—<identifier>s for use with <get diagnostics statement>

<identifier>	Data Type
<statement information item name>s	
All alternatives from ISO/IEC 9075:1992	
<condition information item name>s	
All alternatives from ISO/IEC 9075:1992	
CONDITION_IDENTIFIER	character varying (*L*)
ROUTINE_CATALOG	character varying (*L*)
ROUTINE_SCHEMA	character varying (*L*)
ROUTINE_NAME	character varying (*L*)
SPECIFIC_NAME	character varying (*L*)
PARAMETER_NAME	character varying (*L*)

Access Rules

No additional Access Rules.

General Rules

Table 13—SQL-statement character codes for use in the diagnostics area

SQL-statement	Identifier
All alternatives from ISO/IEC 9075:1992	
<assignment statement>	ASSIGNMENT
<call statement>	CALL
<case statement>	CASE
<compound statement>	BEGIN END
<drop module statement>	DROP MODULE
<drop routine statement>	DROP ROUTINE
<for statement>	FOR
<if statement>	IF
<leave statement>	LEAVE
<loop statement>	LOOP
<repeat statement>	REPEAT
<resignal statement>	RESIGNAL
<return statement>	RETURN
<schema routine>	CREATE ROUTINE
<set path statement>	SET PATH
<signal statement>	SIGNAL
<SQL-server module definition>	CREATE MODULE
<SQL variable declaration>	DECLARE VARIABLE
<while statement>	WHILE

1) Modifies GR3) Specification of <condition information item> retrieves information about the *N*-th condition in the diagnostics area into the <simple target specification>.

   a) Insert before GR3)k) If the value of RETURNED_SQLSTATE corresponds to *external routine invocation exception*, *external routine exception*, *SQL routine exception*, or *warning*, then:

      i) The values of ROUTINE_CATALOG and ROUTINE_SCHEMA are the <catalog name> and the <unqualified schema name>, respectively, of the <schema name> of the schema containing the routine.

      ii) The values of ROUTINE_NAME and SPECIFIC_NAME are the <identifier> of the <routine name> and the <identifier> of the <specific name>, respectively, of the routine.

      iii) The value of PARAMETER_NAME is a zero-length string.

b) Insert before GR3)k) If the value of RETURNED_SQLSTATE corresponds to *data exception — numeric value out of range*, *data exception — string data, right truncation*, *data exception — interval field overflow*, *warning — string data, right truncation*, or *warning — implicit zero-bit padding*, and the condition was raised as a result of an assignment to an SQL parameter during an SQL-invoked routine invocation, then:

i) The values of ROUTINE_CATALOG and ROUTINE_SCHEMA are the <catalog name> and the <unqualified schema name>, respectively, of the <schema name> of the schema containing the routine.

ii) The values of ROUTINE_NAME and SPECIFIC_NAME are the <identifier> of the <routine name> and the <identifier> of the <specific name>, respectively, of the routine.

iii) If the SQL parameter for which the condition was raised specified an SQL parameter name, then the value of PARAMETER_NAME is the SQL parameter name of that SQL parameter; otherwise, the value of PARAMETER_NAME is a zero-length string.

c) Replace GR3)k) If the value of RETURNED_SQLSTATE corresponds to *external routine invocation exception*, *external routine exception*, or *warning*, then the value of MESSAGE_TEXT is the message text item of the routine that raised the condition. Otherwise, the value of MESSAGE_TEXT is an implementation-defined character string.

NOTE 46 – An implementation may set this to <space>s, to a zero-length string, or to a character string describing the condition indicated by RETURNED_SQLSTATE.

d) Insert before GR3)o) If COMMAND_FUNCTION or DYNAMIC_FUNCTION identifies a <signal statement> or <resignal statement>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME, CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, COLUMN_NAME, CURSOR_NAME, are not set as specified in ISO/IEC 9075:1992, but instead are set to a zero-length string.

e) Insert before GR3)o) If the value of the RETURNED_SQLSTATE corresponds to *unhandled user-defined exception*, then the value of CONDITION_IDENTIFIER is the <condition name> of the user-defined exception.

17.2 <signal statement>

Function

Signal an exception condition.

Format

```
<signal statement> ::=
      SIGNAL <signal value>

<signal value> ::=
        <condition name>
      | <sqlstate value>
```

Syntax Rules

1) Case:

   a) If <signal value> immediately contains <condition name>, then:

      i) Let *CN* be the <condition name> contained in the <signal statement>.

      ii) *CN* shall be contained within the scope of one or more <condition name>s whose associated <condition declaration> includes a condition whose <identifier> is *CN*. If there is more than one such <condition name>, then the one with the innermost scope is specified. Let *C* be that condition.

   b) Otherwise, let *C* be the SQLSTATE value defined by <sqlstate value> and let *CN* be a zero-length string.

Access Rules

None.

General Rules

1) Let *N* be the value of the statement information field NUMBER in the diagnostics area before the execution of the <signal statement>. The existing exception information areas 1 through *N* in the diagnostics area are cleared. The value of the statement information field NUMBER in the diagnostics area is set to 1 and the MORE field is set to 'N'.

   The statement information field COMMAND_FUNCTION is set to 'SIGNAL' and the DYNAMIC_FUNCTION field is set to a zero-length string. In the first exception information area in the diagnostics area, the field CONDITION_IDENTIFIER is set to contain *CN*. If *C* has an associated SQLSTATE value, then the exception information field RETURNED_SQLSTATE is set to that value.

2) The following <resignal statement> is effectively executed without further Syntax Rule checking:

   ```
   RESIGNAL
   ```

17.3 <resignal statement>

Function

Resignal an exception condition.

Format

```
<resignal statement> ::=
       RESIGNAL   [ <signal value> ]
```

Syntax Rules

1) Let *RS* be the <resignal statement>.

2) If <signal value> is specified, then

   Case:

   a) If <signal value> immediately contains <condition name>, then:

      i) Let *CN* be the <condition name> contained in *RS*.

      ii) *CN* shall be contained within the scope of one or more <condition name>s whose associated <condition declaration> includes a condition whose <identifier> is *CN*. If there is more than one such <condition name>, then the one with the innermost scope is specified. Let *C* be that condition.

   b) Otherwise, let *C* be the SQLSTATE value defined by <sqlstate value> and let *CN* be a zero-length string.

Access Rules

None.

General Rules

1) Case:

   a) If the first condition in the diagnostics area has no RETURN_SQLSTATE value and the value of the CONDITION_IDENTIFIER is a zero-length string, then an exception condition is raised: *resignal when handler not active*.

   b) Otherwise, let *N* be the value of the statement information field NUMBER in the diagnostics area before the execution of *RS*.

      Case:

      i) If <signal value> is not specified, then the diagnostics area remains unchanged.

      ii) If <signal value> is specified, then the statement information field NUMBER in the diagnostics area is incremented. All existing condition areas are stacked such that the *i*-th condition area is placed at the position of the *i*+1-st condition area in the diagnostics area. If the maximum number of condition areas for the diagnostics area is exceeded,

then the value of the statement information field NUMBER contains the number of exception or completion conditions of the SQL-statement that raised the condition plus those raised by *RS*, and the value of the statement information field MORE is 'Y'.

In the first condition area in the diagnostics area, the statement information field COMMAND_FUNCTION is set to 'RESIGNAL', the DYNAMIC_FUNCTION field is set to a zero-length string, and CONDITION_IDENTIFIER is set to contain *CN*. If *C* has an associated SQLSTATE value, then the condition information field RETURNED_SQLSTATE is set to that value.

2) Case:

a) If the first condition in the diagnostics area has a RETURNED_SQLSTATE value, then:

i) Let *S* be that value.

ii) If a handler *H* is the most appropriate handler for *S*, then *H* is activated.

iii) If no handler is activated and *S* identifies an SQLSTATE value associated with an exception condition, then this is an *unhandled exception condition* and the <SQL procedure statement> that resulted in execution of this <resignal statement> is terminated with this exception condition.

NOTE 47 – If *S* identifies an SQLSTATE value associated with a completion condition, then this is an unhandled condition, and processing continues without altering the flow of control.

b) Otherwise:

i) Let *E* be the value of the CONDITION_IDENTIFIER field of the first condition in the diagnostics area.

ii) If a handler *H* is the most appropriate handler for *E*, then *H* is activated.

iii) If no handler is activated, then this is an *unhandled exception condition* and the <SQL procedure statement> that resulted in execution of *RS* is terminated with the exception condition: *unhandled user-defined exception*.

18 Embedded SQL

18.1 <embedded SQL host program>

Function

Specify an <embedded SQL host program>.

Format

```
<statement or declaration> ::=
         !! All alternatives from ISO/IEC 9075:1992
       | <embedded authorization statement>
       | <embedded path specification>

<embedded authorization statement> ::=
       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>
```

Syntax Rules

1) Insert this SR If an <embedded authorization statement> appears in an <embedded SQL host program>, then it shall be contained in the first <embedded SQL statement> of that <embedded SQL host program>.

2) Insert this SR An <embedded SQL host program> shall not contain more than one <embedded path specification>.

3) Replace SR14)d) Case:

   i) If *H* contains an <embedded authorization statement> *EAS*, then let *EAC* be the <embedded authorization clause> contained in *EAS*; *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*".

4) Insert before SR 14)e) Case:

i) If *H* contains an <embedded path specification> *EPS*, then *M* contains the <module path specification> *EPS*.

ii) Otherwise, *M* contains an implementation-defined <module path specification>.

Access Rules

No additional Access Rules.

General Rules

No additional Access Rules.

19 Information Schema and Definition Schema

19.1 Information Schema

19.1.1 SCHEMATA view

Function

Identify the schemata that are owned by a given user.

Definition

```
CREATE VIEW SCHEMATA
       AS SELECT
           CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER,
           DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA,
           DEFAULT_CHARACTER_SET_NAME, SQL_PATH
         FROM DEFINITION_SCHEMA.SCHEMATA
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND CATALOG_NAME
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.2 DOMAINS view

Function

Identify the domains defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW DOMAINS
       AS SELECT DISTINCT
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         COLLATION_CATALOG, COLLATION_SCHEMA,COLLATION_NAME,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
         DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION, DOMAIN_DEFAULT
       FROM DEFINITION_SCHEMA.DOMAINS
            JOIN
              DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
              LEFT JOIN
              DEFINITION_SCHEMA.COLLATIONS AS S
              USING ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
            ON
              ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, '' ,0 )
              = ( OBJECT_CATALOG, OBJECT_SCHEMA,
                    OBJECT_NAME, COLUMN_NAME , ORDINAL_POSITION ) )
         WHERE
               ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, 'DOMAIN' )
                 IN
                 ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                     WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
               OR
               ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IN
                 ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
                     FROM COLUMNS ) )
           AND DOMAIN_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.3 COLUMNS view

Function

Identify the columns of tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW COLUMNS
       AS SELECT DISTINCT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
         C.COLUMN_NAME, ORDINAL_POSITION,
         CASE WHEN EXISTS ( SELECT *
                FROM DEFINITION_SCHEMA.SCHEMATA AS S
                WHERE ( TABLE_CATALOG, TABLE_SCHEMA )
                    = (S.CATALOG_NAME, S.SCHEMA_NAME )
                  AND SCHEMA_OWNER = USER )
              THEN COLUMN_DEFAULT
             ELSE NULL
         END AS COLUMN_DEFAULT,
         IS_NULLABLE,
         COALESCE (D1.DATA_TYPE, D2.DATA_TYPE) AS DATA_TYPE,
         COALESCE (D1.CHARACTER_MAXIMUM_LENGTH, D2.CHARACTER_MAXIMUM_LENGTH)
           AS CHARACTER_MAXIMUM_LENGTH,
         COALESCE (D1.CHARACTER_OCTET_LENGTH, D2.CHARACTER_OCTET_LENGTH)
                   AS CHARACTER_OCTET_LENGTH,
         COALESCE (D1.NUMERIC_PRECISION, D2.NUMERIC_PRECISION)
                   AS NUMERIC_PRECISION,
         COALESCE (D1.NUMERIC_PRECISION_RADIX, D2.NUMERIC_PRECISION_RADIX)
                   AS NUMERIC_PRECISION_RADIX,
         COALESCE (D1.NUMERIC_SCALE, D2.NUMERIC_SCALE) AS NUMERIC_SCALE,
         COALESCE (D1.DATETIME_PRECISION, D2.DATETIME_PRECISION) AS DATETIME_PRECISION,
         COALESCE (D1.INTERVAL_TYPE, D3.INTERVAL_TYPE) AS INTERVAL_TYPE,
         COALESCE (D1.INTERVAL_PRECISION, D3.INTERVAL_PRECISION) AS INTERVAL_PRECISION,
         COALESCE (C1.CHARACTER_SET_CATALOG, C2.CHARACTER_SET_CATALOG)
                   AS CHARACTER_SET_CATALOG,
         COALESCE (C1.CHARACTER_SET_SCHEMA, C2.CHARACTER_SET_SCHEMA)
                   AS CHARACTER_SET_SCHEMA,
         COALESCE (C1.CHARACTER_SET_NAME, C2.CHARACTER_SET_NAME) AS CHARACTER_SET_NAME,
         COALESCE (D1.COLLATION_CATALOG, D2.COLLATION_CATALOG) AS COLLATION_CATALOG,
         COALESCE (D1.COLLATION_SCHEMA, D2.COLLATION_SCHEMA) AS COLLATION_SCHEMA,
         COALESCE (D1.COLLATION_NAME, D2.COLLATION_NAME) AS COLLATION_NAME,
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
       FROM DEFINITION_SCHEMA.COLUMNS AS C
           LEFT JOIN
             DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
               LEFT JOIN
                 DEFINITION_SCHEMA.COLLATIONS AS C1
                 ON
                 ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA, C1.COLLATION_NAME )
                 = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA, D1.COLLATION_NAME ) )
           ON
             ( ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, 'COLUMN',
                 C.COLUMN_NAME, 0 )
             = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA, D1.OBJECT_NAME,
                   D1.OBJECT_TYPE, D1.COLUMN_NAME, D1.ORDINAL_POSITION ) )
           LEFT JOIN
             DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
               LEFT JOIN
                 DEFINITION_SCHEMA.COLLATIONS AS C2
                 ON
                 ( ( C2.COLLATION_CATALOG, C2.COLLATION_SCHEMA, C2.COLLATION_NAME )
```

```
      = ( D2.COLLATION_CATALOG, D2.COLLATION_SCHEMA, D2.COLLATION_NAME ) )
ON
  ( ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA, C.DOMAIN_NAME, 'DOMAIN',
      '', 0 )
  = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA, D2.OBJECT_NAME,
      D2.OBJECT_TYPE, D2.COLUMN_NAME, D2.ORDINAL_POSITION ) )
WHERE ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME )
  IN
    ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
        FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
          WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
AND C.TABLE_CATALOG
  = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.4 MODULES view

Function

Identify the SQL-server modules in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW MODULES AS
       SELECT
         MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME,
         DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA,
         DEFAULT_CHARACTER_SET_NAME,
         DEFAULT_SCHEMA_CATALOG, DEFAULT_SCHEMA,
         CASE WHEN ( MODULE_CATALOG, MODULE_SCHEMA, CURRENT_USER )
                IN ( SELECT CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER
                     FROM DEFINITION_SCHEMA.SCHEMATA )
           THEN MODULE_DEFINITION
           ELSE NULL
         END AS MODULE_DEFINITION,
         SQL_PATH
       FROM DEFINITION_SCHEMA.MODULES
       WHERE ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME )
               IN ( SELECT MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME
                    FROM DEFINITION_SCHEMA.MODULE_PRIVILEGES
                    WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
             )
         AND MODULE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.5 ROUTINES view

Function

Identify the SQL-invoked routines in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW ROUTINES AS
       SELECT
         SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
         ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE,
         DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
         DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION,
         LANGUAGE, IS_DETERMINISTIC, SQL_DATA_ACCESS, ROUTINE_BODY,
         CASE WHEN ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, CURRENT_USER )
                     IN ( SELECT CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER
                          FROM DEFINITION_SCHEMA.SCHEMATA )
                THEN ROUTINE_DEFINITION
                ELSE NULL
              END AS ROUTINE_DEFINITION,
         EXTERNAL_NAME, PARAMETER_STYLE,
         SQL_PATH, SCHEMA_LEVEL_ROUTINE
       FROM DEFINITION_SCHEMA.ROUTINES R
         LEFT JOIN DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D
           ON
             ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
               'FUNCTION_RESULT', '', 0 )
             = ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
               OBJECT_TYPE, COLUMN_NAME, ORDINAL_POSITION )
       WHERE
         ( ( ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME) IS NULL
         AND
         ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME )
           IN ( SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME
                FROM DEFINITION_SCHEMA.ROUTINE_PRIVILEGES
                WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
           )
         )
       OR
         ( ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME) IS NOT NULL
         AND
         ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME )
           IN ( SELECT MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME
                FROM DEFINITION_SCHEMA.MODULE_PRIVILEGES
                WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
           )
         )
       )
       AND SPECIFIC_CATALOG
           = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.6 PARAMETERS view

Function

Identify the SQL parameters of SQL-invoked routines defined in this catalog.

Definition

```
CREATE VIEW PARAMETERS AS
       SELECT
         P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME,
         P1.ORDINAL_POSITION, PARAMETER_MODE, PARAMETER_NAME,
         DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA, D1.COLLATION_NAME,
         C1.CHARACTER_SET_CATALOG, C1.CHARACTER_SET_SCHEMA,
         C1.CHARACTER_SET_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
         DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION
       FROM DEFINITION_SCHEMA.PARAMETERS P1
         LEFT JOIN
         DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D1
           LEFT JOIN DEFINITION_SCHEMA.COLLATIONS C1
             ON ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA,
                    C1.COLLATION_NAME )
                = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA,
                    D1.COLLATION_NAME ) )
           ON
             (SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
             'PARAMETER', '', P1.ORDINAL_POSITION)
           = (OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
              OBJECT_TYPE, COLUMN_NAME, D1.ORDINAL_POSITION)
           INNER JOIN DEFINITION_SCHEMA.ROUTINES R1
             ON ( ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME )
              = ( R1.SPECIFIC_CATALOG, R1.SPECIFIC_SCHEMA, R1.SPECIFIC_NAME )
         )
       WHERE
         ( ( ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME) IS NULL
         AND
         ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME )
           IN (
           SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME
           FROM DEFINITION_SCHEMA.ROUTINE_PRIVILEGES
           WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
           )
         )
         OR
           ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME) IS NOT NULL
           AND
           ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME )
           IN (
             SELECT MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME
             FROM DEFINITION_SCHEMA.MODULE_PRIVILEGES
             WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
             )
           )
         )
         AND P1.SPECIFIC_CATALOG
           = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.7 MODULE_TABLE_USAGE view

Function

Identify the tables owned by a given user on which SQL-server modules defined in this catalog are dependent.

Definition

```
CREATE VIEW MODULE_TABLE_USAGE
       AS
            MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME,
            TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
          FROM DEFINITION_SCHEMA.MODULE_TABLE_USAGE
               JOIN
               DEFINITION_SCHEMA.SCHEMATA S
               ON
                  ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
                    ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
          WHERE SCHEMA_OWNER = CURRENT_USER
            AND MODULE_CATALOG = ( SELECT CATALOG_NAME
                                   FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.8 MODULE_COLUMN_USAGE view

Function

Identify the columns owned by a given user on which SQL-server modules defined in this catalog are dependent.

Definition

```
CREATE VIEW MODULE_COLUMN_USAGE
       AS SELECT
            ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME,
            TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
          FROM DEFINITION_SCHEMA.MODULE_COLUMN_USAGE
               JOIN DEFINITION_SCHEMA.SCHEMATA S
                 ON ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
                        ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
          WHERE SCHEMA_OWNER = CURRENT_USER
            AND MODULE_CATALOG = ( SELECT CATALOG_NAME
                                   FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.9 MODULE_PRIVILEGES view

Function

Identify the privileges on SQL-server modules defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW MODULE_PRIVILEGES AS
       SELECT
         GRANTOR, GRANTEE, MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME,
         PRIVILEGE_TYPE, IS_GRANTABLE
       FROM DEFINITION_SCHEMA.MODULE_PRIVILEGES
       WHERE     ( GRANTEE IN ( 'PUBLIC', CURRENT_USER ) OR
                   GRANTEE = CURRENT_USER )
            AND MODULE_CATALOG =
                ( SELECT CATALOG_NAME
                  FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

19.1.10 ROUTINE_PRIVILEGES view

Function

Identify the privileges on SQL-invoked routines defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW ROUTINE_PRIVILEGES AS
       SELECT
         GRANTOR, GRANTEE, SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
         ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME,
         PRIVILEGE_TYPE, IS_GRANTABLE
       FROM ( SELECT  GRANTOR, GRANTEE, SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,

                      ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME,
                      PRIVILEGE_TYPE, IS_GRANTABLE
              FROM DEFINITION_SCHEMA.ROUTINE_PRIVILEGES
              WHERE     ( GRANTEE IN ( 'PUBLIC', CURRENT_USER ) OR
                          GRANTEE = CURRENT_USER )
                    AND ROUTINE_CATALOG =
                          ( SELECT CATALOG_NAME
                            FROM INFORMATION_SCHEMA_CATALOG_NAME )
              AS RP
       JOIN DEFINITION_SCHEMA.ROUTINES
         USING ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME )
```

19.1.11 ROUTINE_TABLE_USAGE view

Function

Identify the tables owned by a given user on which SQL-invoked routines defined in this catalog are dependent.

Definition

```
CREATE VIEW ROUTINE_TABLE_USAGE AS
       SELECT
         SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
         ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME,
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
       FROM DEFINITION_SCHEMA.ROUTINE_TABLE_USAGE
       JOIN DEFINITION_SCHEMA.ROUTINES
         USING ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME )
       JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
              ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
       WHERE SCHEMA_OWNER = CURRENT_USER
         AND SPECIFIC_CATALOG =
           ( SELECT CATALOG_NAME
             FROM INFORMATION_SCHEMA_CATALOG_NAME )
```