Final Committee Draft ISO/IEC FCD	
Date: **1999-11-30**	Reference number: ISO/JTC 1/SC 32 **N 0368**
Supersedes document SC 32 N 0197	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange Secretariat: USA (ANSI)	Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by: **2000-05-01** Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.

ISO/IEC FCD 9075-9

Title: Information Technology - Database Language SQL - Part 9: Management of External Data

Project: 1.03.04.09.00.00

Introductory note: The attached document is hereby submitted for a four-month letter ballot to the National Bodies of ISO/IEC JTC 1/SC 32. The ballot starts 2000-01-01; ITTF must have the document two weeks before the balloting can start.

Medium: E

No. of pages: 397

Address Reply to: Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32, Pacific Northwest National Laboratory, 901 D Street, SW., Suite 900, Washington, DC, 20024-2115, United States of America

Telephone: +1 703 575 2114; Facsimile; +1 703 681 9180; E-mail: MannD@battelle.org

X3H2-99-472
WG3:SAF-017
ISO/IEC JTC 1/SC 32 N00368

FCD 9075-9

Database Language SQL — Part 9: SQL/MED

«Part 9»

November 1999

Contents

TABLES

Tables **Page**

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, Part 9, was prepared by Joint Technical Committee ISO/IEC JTC 1, Information technology.

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

— Part 1: Framework (SQL/Framework)

— Part 2: Foundation (SQL/Foundation)

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

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

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

— Part 7: Temporal (SQL/Temporal)

— Part 9: Management of External Data (SQL/MED)

— Part 10: Object Language Bindings (SQL/OLB)

— Part 11: Information and Definition Schemas (SQL/Schemata)

Introduction

The organization of this Part of this International Standard 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 related to this part of ISO/IEC 9075.

5) Clause 5, "Lexical elements", defines the lexical elements of the language specified in this part of ISO/IEC 9075.

6) Clause 6, "Scalar expressions", defines the elements of the language that produce scalar values.

7) Clause 7, "Query expressions", defines the elements of the language that produce rows and tables of data.

8) Clause 8, "Predicates", defines the predicates of the language.

9) Clause 9, "URLs", specifies the format of URLs.

10) Clause 10, "Data assignment rules and routine determination", specifies the rules for assignments that retrieves data from or store data into SQL-data, and formation rules for set operations.

11) Clause 11, "Additional common elements", defines additional common elements used in the definition of foreign tables, foreign servers, and foreign-data wrappers.

12) Clause 12, "Schema definition and manipulation", defines facilities related to foreign tables and datalink data type support for creating and managing a schema.

13) Clause 13, "Foreign server and foreign-data wrapper definition and manipulation", defines facilities related to creating and managing foreign servers and foreign-data wrappers.

14) Clause 14, "Access control", defines facilities for controlling access to SQL-data.

15) Clause 15, "SQL-client modules", defines SQL-client modules and externally-invoked procedures.

16) Clause 16, "Data manipulation", defines the data manipulation statements.

17) Clause 17, "Dynamic SQL", defines the dynamic SQL statements.

18) Clause 18, "Embedded SQL", defines the embedded SQL statements.

19) Clause 19, "Call-Level Interface specifications", defines facilities for using SQL through a Call-Level Interface.

20) Clause 20, "SQL/CLI routines", defines each of the routines that comprise the Call-Level Interface.

21) Clause 22, "Foreign-data wrapper interface routines", specifies the interaction between an SQL-server and a foreign-data wrapper.

22) Clause 23, "Information Schema", defines viewed tables that contain schema information.

23) Clause 24, "Definition Schema", defines base tables on which the viewed tables containing schema information depend.

24) Clause 25, "Status codes", defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

25) Clause 26, "Conformance", specifies the way in which conformance to this part of ISO/IEC 9075 may be claimed.

26) Annex A, "SQL Conformance Summary", is an informative Annex. It summarizes the conformance requirements of the SQL language.

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

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

29) Annex D, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this part of ISO/IEC 9075.

30) Annex E, "Incompatibilities with ISO/IEC 9075:1999", is an informative Annex. It lists incompatibilities with the previous version of ISO/IEC 9075.

31) Annex F, "Typical header files", is an informative Annex. It provides examples of typical definition files for application programs using the SQL Call-Level Interface.

32) Annex G, "SQL feature and package taxonomy", is an informative Annex. It identifies features and packages of the SQL language specified in this part of ISO/IEC 9075 by an identifier and a short descriptive name. This taxonomy is used to specify conformance to the packages specified in this part of ISO/IEC 9075. The feature taxonomy may be used to develop other profiles involving the SQL language.

In the text of this part of ISO/IEC 9075, Clauses begin a new odd-numbered page. Any resulting blank space is not significant.

Information technology — Database languages — SQL — Part 9: SQL/MED

****Editor's Note****
We still must INDEX the names of all routines and all parameters to all routines.

1 Scope

This part of ISO/IEC 9075 defines extensions to Database Language SQL to support management of external data through the use of foreign tables and datalink data types.

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

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 listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

2.1 ISO/IEC JTC 1 standards

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

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

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

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

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

— ISO/IEC 9075-1:1999, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).*

— ISO/IEC 9075-2:1999, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).*

— ISO/IEC 9075-3:1999, *Information technology — Database languages — SQL — Part 3: Call-level interface (SQL/CLI).*

— ISO/IEC 9075-4:1999, *Information technology — Database languages — SQL — Part 4: Persistent stored modules (SQL/PSM).*

— ISO/IEC 9075-5:1999, *Information technology — Database languages — SQL — Part 5: Host language bindings (SQL/Bindings).*

— 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*.

2.2 Publicly-available specifications

— RFC 1738: *Uniform Resource Locators (URL)*, T. Berners-Lee, L. Masinter, M. McCahill, December, 1994.

— RFC 1808, *Relative Uniform Resource Locators*, R. Fielding, June, 1995.

— RFC 2638, *The mailto URL scheme*, R. Hoffman, L. Masinter, J. Zawinski, July, 1998.

3 Definitions, notations, and conventions

3.1 Definitions

3.1.1 Definitions provided in Part 9

Insert this paragraph For the purposes of this part of ISO/IEC 9075, the definitions given in ISO/IEC 9075-1, ISO/IEC 9075-2, ISO/IEC 9075-3, ISO/IEC 9075-4, and ISO/IEC 9075-5 and the following definitions apply.

- 2 definitions deleted.

a) **access token**: An encrypted value returned under certain conditions by an SQL-server as integral part of the File Reference of a datalink value.

b) **datalink**: A value, of data type DATALINK, representing some external data source (*i.e.*, one that is not part of the SQL-environment). The data source is assumed to be accessible through some external data manager.

c) **datalinker**: An implementation-dependent component for controlling access and referential integrity to external data sources.

d) **external data**: Data that is not managed by an SQL-server involved in an SQL-session, but that can be a source of SQL-data.

e) **foreign-data wrapper**: A named collection of routines, invocable by the SQL-server, supporting the programming interface specified for such routines in this part of ISO/IEC 9075.

f) **foreign server**: A named agency, external to the SQL-environment, but known to the SQL-server, that manages external data.

g) **foreign table**: A named base table whose rows are supplied when needed by some foreign server. The mechanism by which these rows are supplied is provided by a foreign-data wrapper.

h) **integrity option**: Specifies the level of integrity of the link between a datalink and the actual object.

i) **link control**: A property of a column of data type DATALINK, specifying the extent to which the links between datalinks in that column and the external data sources they reference are to be monitored (in various specific sense).

j) **read permission option**: A link control option specifying how permission to read external data sources referenced by certain datalinks is determined.

k) **recovery option**: A link control option specifying whether or not point in time recovery is required for the external data sources referenced by certain datalinks.

l) **unlink option**: A link control option specifying the action to be taken when certain sites occupied by datalinks are updated or deleted.

m) **user mapping**: An implementation-defined mapping of an authorization identifier to an equivalent concept maintained by a foreign server. .

n) **write permission option**: A link control option specifying how permission to write external data sources referenced by certain datalinks is determined.

3.2 Notations

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"). This version of BNF is fully described in Subclause 6.1, "Notation", of ISO/IEC 9075-1.

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 ISO/IEC 9075-1 and ISO/IEC 9075-2.

3.3.1 Relationships to other parts of ISO/IEC 9075

3.3.1.1 Clause, Subclause, and Table relationships

Table 1—Clause, Subclause, and Table relationships

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Clause 1, "Scope"	Clause 1, "Scope"	ISO/IEC 9075-2
Clause 2, "Normative references"	Clause 2, "Normative references"	ISO/IEC 9075-2
Clause 3, "Definitions, notations, and conventions"	Clause 3, "Definitions, notations, and conventions"	ISO/IEC 9075-2
Subclause 3.1, "Definitions"	Subcluase 3.1, "Definitions"	ISO/IEC 9075-2
Subclause 3.1.1, "Definitions provided in Part 9"	*none*	*none*
Subclause 3.2, "Notations"	Subclause 6.1, "Notation"	ISO/IEC 9075-1
Subclause 3.3, "Conventions"	Subclause 3.3, "Conventions"	ISO/IEC 9075-2
Subclause 3.3.1, "Relationships to other parts of ISO/IEC 9075"	*none*	*none*
Subclause 3.3.1.1, "Clause, Subclause, and Table relationships"	*none*	*none*
Subclause 3.4, "Object identifier for Database Language SQL"	Subclause 6.3, "Object identifier for Database Language SQL"	ISO/IEC 9075-1
Clause 4, "Concepts"	Clause 4, "Concepts"	ISO/IEC 9075-2

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 4.1, “Data types”	Subclause 4.1, "Data types"	ISO/IEC 9075-2
Subclause 4.2, “Foreign servers”	*none*	*none*
Subclause 4.3, “Foreign-data wrappers”	*none*	*none*
Subclause 4.4, “User mappings”	*none*	*none*
Subclause 4.5, “Generic options”	*none*	*none*
• 1 row deleted.		
Subclause 4.7, “Datalinks”	*none*	*none*
Subclause 4.7.1, “Operations involving datalinks”	*none*	*none*
Subclause 4.7.1.1, “Operators that operate on datalinks”	*none*	*none*
Subclause 4.7.1.2, “Other operators involving datalinks”	*none*	*none*
Subclause 4.8, “Type conversion and mixing of data types”	Subclause 4.12, "Type conversions and mixing of data types"	ISO/IEC 9075-2
Subclause 4.9, “Columns, fields, and attributes”	Subclause 4.15, "Columns, fields, and attributes"	ISO/IEC 9075-2
Subclause 4.10, “Tables”	Subclause 4.16, "Tables"	ISO/IEC 9075-2
Subclause 4.11, “SQL-statements”	4.30, "SQL-statements"	ISO/IEC 9075-2
Subclause 4.11.1, “SQL-statements classified by function”	Subclause 4.30.2, "SQL-statements classified by function"	ISO/IEC 9075-2
Subclause 4.12, “SQL-sessions”	Subclause 4.34, "SQL-sessions"	ISO/IEC 9075-3
Subclause 4.13, “Privileges”	Subclause 4.31.2, "Privileges"	ISO/IEC 9075-2
Subclause 4.14, “Foreign-data wrapper interface”	*none*	*none*
Subclause 4.14.1, “Handles”	*none*	*none*
Subclause 4.14.2, “Foreign-data wrapper interface routines”	*none*	*none*
Subclause 4.14.2.1, “Handle routines”	*none*	*none*
Subclause 4.14.2.2, “Initialization routines”	*none*	*none*
Subclause 4.14.2.3, “Access routines”	*none*	*none*
Subclause 4.14.2.4, “Termination routines”	*none*	*none*

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 4.14.3, "Return codes"	*none*	*none*
Subclause 4.14.5, "Null terminated strings"	*none*	*none*
Subclause 4.14.6, "Null pointers"	*none*	*none*
Subclause 4.15, "Introduction to SQL/CLI"	Subclause 4.1, "Introduction to SQL/CLI"	ISO/IEC 9075-3
Clause 5, "Lexical elements"	Clause 4, "Lexical elements"	ISO/IEC 9075-2
Subclause 5.1, "<SQL terminal character>"	Subclause 5.1, "<SQL terminal character>"	ISO/IEC 9075-2
Subclause 5.2, "<token> and <separator>"	Subclause 5.2, "<token> and <separator>"	ISO/IEC 9075-2
Subclause 5.3, "Names and identifiers"	Subclause 5.4, "Names and identifiers"	ISO/IEC 9075-2
Clause 6, "Scalar expressions"	Clause 6, "Scalar expressions"	ISO/IEC 9075-2
Subclause 6.1, "<data type>"	Subclause 6.1, "<data type>"	ISO/IEC 9075-2
Subclause 6.2, "<set function specification>"	Subclause 6.16, "<set function specification>"	ISO/IEC 9075-2
Subclause 6.3, "<string value function>"	Subclause 6.18, "<string value function>"	ISO/IEC 9075-2
Subclause 6.4, "<datalink value function>"	*none*	*none*
Subclause 6.5, "<cast specification>"	Subclause 6.22, "<cast specification>"	ISO/IEC 9075-2
Subclause 6.6, "<value expression>"	Subclause 6.23, "<value expression>"	ISO/IEC 9075-2
Subclause 6.7, "<datalink value expression>"	*none*	*none*
Clause 7, "Query expressions"	Clause 7, "Query expressions"	ISO/IEC 9075-2
Subclause 7.1, "<table reference>"	Subclause 7.6, "<table reference>"	ISO/IEC 9075-2
Subclause 7.2, "<joined table>"	Subclause 7.7, "<joined table>"	ISO/IEC 9075-2
Subclause 7.3, "<group by clause>"	Subclause 7.8, "<group by clause>"	ISO/IEC 9075-2
Subclause 7.4, "<query expression>"	Subclause 7.12, "<query expression>"	ISO/IEC 9075-2
Clause 8, "Predicates"	Clause 8, "Predicates"	ISO/IEC 9075-2
Subclause 8.1, "<comparison predicate>"	Subclause 8.2, "<comparison predicate>"	ISO/IEC 9075-2
Clause 9, "URLs"	*none*	*none*

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 9.1, “URL format”	*none*	*none*
Clause 10, “Data assignment rules and routine determination”	Clause 9, "Data assignment rules and routine determination"	ISO/IEC 9075-2
Subclause 10.1, “Retrieval assignment”	Subclause 9.1, "Retrieval assignment"	ISO/IEC 9075-2
Subclause 10.2, “Store assignment”	Subclause 9.2, "Store assignment"	ISO/IEC 9075-2
Subclause 10.3, “Data types of results of aggregations”	Subclause 9.3, "Data types of results of aggregations"	ISO/IEC 9075-2
Subclause 10.4, “Type precedence list determination”	Subclause 9.5, "Type precedence list determination"	ISO/IEC 9075-2
Clause 11, “Additional common elements”	Clause 10, "Additional common elements"	ISO/IEC 9075-2
Subclause 11.1, “<privileges>”	Subclaue 4.31.2, "Privileges"	ISO/IEC 9075-2
Subclause 11.2, “<generic options>”	*none*	*none*
Subclause 11.3, “<alter generic options>”	*none*	*none*
Clause 12, “Schema definition and manipulation”	Clauase 11, "Schema definition and manipulation"	ISO/IEC 9075-2
Subclause 12.1, “<schema definition>”	Subclause 11.1, "<schema definition>"	ISO/IEC 9075-2
Subclause 12.2, “<table definition>”	Subclause 11.3, "<table definition>"	ISO/IEC 9075-2
Subclause 12.3, “<column definition>”	Subclause 11.4, "<column definition>"	ISO/IEC 9075-2
Subclause 12.4, “<unique constraint definition>”	Subclause 11.7, "<unique constraint definition>"	ISO/IEC 9075-2
Subclause 12.5, “<drop column definition>”	Subclause 11.17, "<drop column definition>"	ISO/IEC 9075-2
• 2 rows deleted.		
Subclause 12.6, “<foreign table definition>”	*none*	*none*
Subclause 12.7, “<alter foreign table statement>”	*none*	*none*
Subclause 12.8, “<add basic column definition>”	*none*	*none*
Subclause 12.10, “<drop basic column definition>”	*none*	*none*
Subclause 12.11, “<drop foreign table statement>”	*none*	*none*

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Clause 13, "Foreign server and foreign-data wrapper definition and manipulation"	*none*	*none*
Subclause 13.1, "<foreign server definition>"	*none*	*none*
Subclause 13.2, "<alter foreign server statement>"	*none*	*none*
Subclause 13.3, "<drop foreign server statement>"	*none*	*none*
Subclause 13.4, "<foreign-data wrapper definition>"	*none*	*none*
Subclause 13.5, "<alter foreign-data wrapper statement>"	*none*	*none*
Subclause 13.6, "<drop foreign-data wrapper statement>"	*none*	*none*
Clause 14, "Access control"	Clause 12, "Access control"	ISO/IEC 9075-2
Subclause 14.2, "<user mapping definition>"	*none*	*none*
Subclause 14.3, "<alter user mapping statement>"	*none*	*none*
Subclause 14.4, "<drop user mapping statement>"	*none*	*none*
Clause 15, "SQL-client modules"	Clause 13, "SQL-client modules"	ISO/IEC 9075-2
Subclause 15.1, "<SQL-client module definition>"	Subclause 13.1, "<SQL-client module definition>"	ISO/IEC 9075-2
Subclause 15.2, "Calls to an <externally-invoked procedure>"	Subclause 13.4, "Calls to an <externally-invoked procedure>"	ISO/IEC 9075-2
Subclause 15.3, "<SQL procedure statement>"	Subclause 13.5, "<SQL procedure statement>"	ISO/IEC 9075-2
Subclause 15.4, "Data type correspondences"	Subclause 13.6, "Data type correspondences"	ISO/IEC 9075-2
Clause 16, "Data manipulation"	Clause 14, "Data manipulation"	ISO/IEC 9075-2
Subclause 16.1, "Effect of deleting rows from base tables"	Subclause 14.14, "Effect of deleting rows from base tables"	ISO/IEC 9075-2
Subclause 16.2, "Effect of inserting tables into base tables"	Subclause 14.17, "Effect of inserting tables into base tables"	ISO/IEC 9075-2
Subclause 16.3, "Effect of replacing rows in base tables"	Subclause 14.20, "Effect of replacing rows in base tables"	ISO/IEC 9075-2
Clause 17, "Dynamic SQL"	Clause 15, "Dynamic SQL"	ISO/IEC 9075-5

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 17.1, "Description of SQL descriptor areas"	Subclause 15.1, "Description of SQL descriptor areas"	ISO/IEC 9075-5
Subclause 17.2, "<describe statement>"	Subclause 15.8, "<describe statement>"	ISO/IEC 9075-5
Clause 18, "Embedded SQL"	Clause 16, "Embedded SQL"	ISO/IEC 9075-5
Subclause 18.1, "<embedded SQL Ada program>"	Subclause 16.3, "<embedded SQL Ada program>"	ISO/IEC 9075-5
Subclause 18.2, "<embedded SQL C program>"	Subclause 16.4, "<embedded SQL C program>"	ISO/IEC 9075-5
Subclause 18.3, "<embedded SQL COBOL program>"	Subclause 16.5, "<embedded SQL COBOL program>"	ISO/IEC 9075-5
Subclause 18.4, "<embedded SQL Fortran program>"	Subclause 16.6, "<embedded SQL Fortran program>"	ISO/IEC 9075-5
Subclause 18.5, "<embedded SQL MUMPS program>"	Subclause 16.7, "<embedded SQL MUMPS program>"	ISO/IEC 9075-5
Subclause 18.6, "<embedded SQL Pascal program>"	Subclause 16.8, "<embedded SQL Pascal program>"	ISO/IEC 9075-5
Subclause 18.7, "<embedded SQL PL/I program>"	Subclause 16.9, "<embedded SQL PL/I program>"	ISO/IEC 9075-5
Clause 19, "Call-Level Interface specifications"	Clause 5, "Call-Level Interface specifications	ISO/IEC 9075-3
Subclause 19.1, "<CLI routine>"	Subclause 5.1, "<CLI routine>"	ISO/IEC 9075-3
Subclause 19.2, "Implicit DESCRIBE USING clause"	Subclause 5.5, "Implicit DESCRIBE USING clause"	ISO/IEC 9075-3
Subclause 19.2.1, "CLI-specific status codes"	Subclause 5.12, "CLI-specific status codes"	ISO/IEC 9075-3
Subclause 19.2.2, "Description of CLI item descriptor areas"	Subclause 5.13, "Description of CLI item descriptor areas"	ISO/IEC 9075-3
Subclause 19.2.3, "Other tables associated with CLI"	Subclause 5.14, "Other tables associated with CLI"	ISO/IEC 9075-3
Subclause 19.3, "SQL/CLI data type correspondences"	Subclause 5.14, "SQL/CLI data type correspondences"	ISO/IEC 9075-3
Clause 20, "SQL/CLI routines"	Clause 6, "SQL/CLI routines"	ISO/IEC 9075-3
Subclause 20.1, "BuildDataLink"	*none*	*none*
Subclause 20.2, "GetDataLinkAttr"	*none*	*none*
Subclause 20.3, "GetInfo"	Subclause 6.38, "GetInfo"	ISO/IEC 9075-3
Clause 21, "SQL/MED common specifications"	*none*	*none*

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 21.3, "Implicit DESCRIBE INPUT USING clause"	*none*	*none*
Subclause 21.4, "Implicit DESCRIBE OUTPUT USING clause"	*none*	*none*
Subclause 21.8, "Character string retrieval"	*none*	*none*
Subclause 21.9, "Binary large object string retrieval"	*none*	*none*
Subclause 21.10, "Deferred parameter check"	*none*	*none*
Subclause 21.11, "MED-specific status codes"	*none*	*none*
Subclause 21.12, "Tables used with SQL/MED"	*none*	*none*
Clause 22, "Foreign-data wrapper interface routines"	*none*	*none*
Subclause 22.1, "<foreign-data wrapper routine>"	*none*	*none*
Subclause 22.2, "<foreign-data wrapper routine> invocation"	*none*	*none*
Subclause 22.4, "AllocWrapperEnv"	*none*	*none*
Subclause 22.6, "ConnectForeignServer"	*none*	*none*

• 1 row deleted.

Subclause 22.9, "FreeFSConnection"	*none*	*none*
Subclause 22.11, "FreeWrapperEnv"	*none*	*none*

• 5 rows deleted.

Subclause 22.30, "GetNumberOfReplySelectListElements"	*none*	*none*
Subclause 22.31, "GetNumberOfReplyTableReferences"	*none*	*none*
Subclause 22.32, "GetNumberOfSelectListElements"	*none*	*none*
Subclause 22.33, "GetNumberOfServerOptions"	*none*	*none*
Subclause 22.34, "GetNumberOfTableReferenceElements"	*none*	*none*
Subclause 22.38, "GetReplySelectListElement"	*none*	*none*
Subclause 22.39, "GetReplyTableReference"	*none*	*none*
Subclause 22.40, "GetSelectListElement"	*none*	*none*
Subclause 22.41, "GetSelectListElementType"	*none*	*none*
Subclause 22.42, "GetServerName"	*none*	*none*

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 22.43, "GetServerOption"	*none*	*none*
Subclause 22.44, "GetServerOptionValueByName"	*none*	*none*
Subclause 22.45, "GetServerType"	*none*	*none*
Subclause 22.46, "GetServerVersion"	*none*	*none*
Subclause 22.52, "GetTableReferenceElement"	*none*	*none*
Subclause 22.53, "GetTableReferenceElementType"	*none*	*none*
Subclause 22.54, "GetTableReferenceTableName"	*none*	*none*
Subclause 22.57, "GetValueExpressionColumnName"	*none*	*none*
Subclause 22.62, "InitForeignRequest"	*none*	*none*
Subclause 22.63, "Iterate"	*none*	*none*
Clause 23, "Information Schema"	Clause 20, "Information Schema"	ISO/IEC 9075-2
Subclause 23.1, "COLUMN_ OPTIONS view"	*none*	*none*
Subclause 23.2, "COLUMNS view"	Subclause 20.18, "COLUMNS view"	ISO/IEC 9075-2
Subclause 23.3, "FOREIGN_DATA_ WRAPPER_OPTIONS view"	*none*	*none*
Subclause 23.4, "FOREIGN_DATA_ WRAPPERS view"	*none*	*none*
Subclause 23.5, "FOREIGN_ SERVER_OPTIONS view"	*none*	*none*
Subclause 23.6, "FOREIGN_ SERVERS view"	*none*	*none*
Subclause 23.7, "FOREIGN_ TABLE_OPTIONS view"	*none*	*none*
Subclause 23.8, "FOREIGN_ TABLES view"	*none*	*none*
Subclause 23.9, "USER_MAP_ OPTIONS view"	*none*	*none*
Subclause 23.10, "USER_ MAPPINGS view"	*none*	*none*
Subclause 23.11, "Short name views"	Subclause 20.69, "Short name views"	ISO/IEC 9075-2
Clause 24, "Definition Schema"	Clause 21, "Definition Schema"	ISO/IEC 9075-2
Subclause 24.1, "COLUMN_ OPTIONS base table"	*none*	*none*
Subclause 24.2, "COLUMNS base table"	Subclause 21.14, "COLUMNS base table"	ISO/IEC 9075-2

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 24.3, "DATA_TYPE_ DESCRIPTOR base table"	Subclause 21.15, "DATA_TYPE_ DESCRIPTOR base table"	ISO/IEC 9075-2
Subclause 24.4, "FOREIGN_DATA_ WRAPPER_OPTIONS base table"	*none*	*none*
Subclause 24.5, "FOREIGN_DATA_ WRAPPERS base table"	*none*	*none*
Subclause 24.6, "FOREIGN_ SERVER_OPTIONS base table"	*none*	*none*
Subclause 24.7, "FOREIGN_ SERVERS base table"	*none*	*none*
Subclause 24.8, "FOREIGN_ TABLE_OPTIONS base table"	*none*	*none*
Subclause 24.9, "FOREIGN_ TABLES base table"	*none*	*none*
Subclause 24.10, "SQL_SIZING base table"	Subclause 7.2, "SQL_SIZING base table"	ISO/IEC 9075-3
Subclause 24.11, "TABLES base table"	Subclause 21.43, "TABLES base table"	ISO/IEC 9075-2
Subclause 24.12, "USAGE_ PRIVILEGES base table"	Subclause 21.50, "USAGE_ PRIVILEGES base table"	ISO/IEC 9075-2
Subclause 24.13, "USER_ MAPPING_OPTIONS base table"	*none*	*none*
Subclause 24.14, "USER_ MAPPINGS base table"	*none*	*none*
Clause 25, "Status codes"	Clause 22, "Status codes"	ISO/IEC 9075-2
Subclause 25.1, "SQLSTATE"	Subclause 22.1, "SQLSTATE"	ISO/IEC 9075-2
Clause 26, "Conformance"	Clause 21, "Conformance"	ISO/IEC 9075-5
Annex A, "SQL Conformance Summary"	Annex A, "SQL Conformance Summary"	ISO/IEC 9075-5
Annex B, "Implementation-defined elements"	Annex B, "Implementation-defined elements"	ISO/IEC 9075-2
Annex C, "Implementation-dependent elements"	Annex C, "Implementation-dependent elements"	ISO/IEC 9075-2
Annex D, "Deprecated features"	Annex D, "Deprecated features"	ISO/IEC 9075-2
Annex E, "Incompatibilities with ISO/IEC 9075:1999"	Annex E, "Incompatibilities with ISO/IEC 9075:1999"	ISO/IEC 9075-2
Annex F, "Typical header files"	Annex A, "Typical header files"	ISO/IEC 9075-3

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause F.1, "C Header File SQLCLI.H"	Subclause A.1, "C header file SQLCLI.H"	ISO/IEC 9075-3
Subclause F.2, "COBOL Library Item SQLCLI"	Subclause A.2, "COBOL library item SQLCLI"	ISO/IEC 9075-3
Annex G, "SQL feature and package taxonomy"	Annex F, "SQL feature and package taxonomy"	ISO/IEC 9075-2
Table 1, "Clause, Subclause, and Table relationships"	*none*	*none*
Table 2, "Valid datalink file control options"	*none*	*none*
Table 13, "Codes used for SQL data types in Dynamic SQL"	Table 4, "Codes used for SQL data types in Dynamic SQL"	ISO/IEC 9075-5
Table 6, "Data type correspondences for Ada"	Table 18, "Data type correspondences for Ada"	ISO/IEC 9075-2
Table 7, "Data type correspondences for C"	Table 19, "Data type correspondences for C"	ISO/IEC 9075-2
Table 8, "Data type correspondences for COBOL"	Table 20, "Data type correspondences for COBOL"	ISO/IEC 9075-2
Table 9, "Data type correspondences for Fortran"	Table 21, "Data type correspondences for Fortran"	ISO/IEC 9075-2
Table 10, "Data type correspondences for MUMPS"	Table 22, "Data type correspondences for MUMPS"	ISO/IEC 9075-2
Table 11, "Data type correspondences for Pascal"	Table 23, "Data type correspondences for Pascal"	ISO/IEC 9075-2
Table 12, "Data type correspondences for PL/I"	Table 24, "Data type correspondences for PL/I"	ISO/IEC 9075-2
Table 14, "Abbreviated SQL/CLI generic names"	Table 4, "Abbreviated SQL/CLI generic names"	ISO/IEC 9075-3
Table 15, "SQLSTATE class and subclass values for SQL/CLI-specific conditions"	Table 5, "SQLSTATE class and subclass values for SQL/CLI-specific conditions"	ISO/IEC 9075-3
Table 16, "Codes used for implementation data types in SQL/CLI"	Table 7, "Codes for implementation data types in SQL/CLI"	ISO/IEC 9075-3
Table 17, "Codes used for application data types in SQL/CLI"	Table 8, "Codes for application data types in SQL/CLI"	ISO/IEC 9075-3
Table 18, "Codes used to identify SQL/CLI routines"	Table 27, "Codes used to identify SQL/CLI routines"	ISO/IEC 9075-3
Table 19, "Codes and data types for implementation information"	Table 28, "Codes and data types for implementation information"	ISO/IEC 9075-3

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Table 20, "Codes used for datalink attributes"	*none*	*none*
Table 21, "Data types of attributes"	Table 19, "Data types of attributes"	ISO/IEC 9075-3
Table 22, "SQL/CLI data type correspondences for Ada"	Table 44, "SQL/CLI data type correspondences for Ada"	ISO/IEC 9075-3
Table 23, "SQL/CLI data type correspondences for C"	Table 45, "SQL/CLI data type correspondences for C"	ISO/IEC 9075-3
Table 24, "SQL/CLI data type correspondences for COBOL"	Table 46, "SQL/CLI data type correspondences for COBOL"	ISO/IEC 9075-3
Table 25, "SQL/CLI data type correspondences for Fortran"	Table 47, "SQL/CLI data type correspondences for Fortran"	ISO/IEC 9075-3
Table 26, "SQL/CLI data type correspondences for MUMPS"	Table 48, "SQL/CLI data type correspondences for MUMPS"	ISO/IEC 9075-3
Table 27, "SQL/CLI data type correspondences for Pascal"	Table 49, "SQL/CLI data type correspondences for Pascal"	ISO/IEC 9075-3
Table 28, "SQL/CLI data type correspondences for PL/I"	Table 50, "SQL/CLI data type correspondences for PL/I"	ISO/IEC 9075-3
Table 29, "SQLSTATE class and subclass values for SQL/MED-specific conditions"	*none*	*none*
Table 30, "Codes used for <table reference> types"	*none*	*none*
Table 31, "Codes used for <value expression> types"	*none*	*none*
Table 38, "SQLSTATE class and subclass values"	Table 27, "SQLState class and subclass values"	ISO/IEC 9075-2
Table 40, "SQL/MED feature taxonomy for features outside Core SQL"	*none*	*none*

3.4 Object identifier for Database Language SQL

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

Format

```
<Part 9 yes> ::=
      <Part 9 conformance> <Part 9 datalink> <Part 9 wrapper>

<Part 9 conformance> ::= 9 | sqlmed200x <left paren> 9 <right paren>

<Part 9 datalink> ::=
```

```
        <Part 9 datalink yes>
      | <Part 9 datalink no>

<Part 9 datalink yes> ::=
      1 | datalinkyes <left paren> 1 <right paren>

<Part 9 datalink no> ::=
      0 | datalinkno <left paren> 0 <right paren>

<Part 9 wrapper> ::=
        <Part 9 wrapper yes>
      | <Part 9 wrapper no>

<Part 9 wrapper yes> ::=
      1 | wrapperyes <left paren> 1 <right paren>

<Part 9 wrapper no> ::=
      0 | wrapperno <left paren> 0 <right paren>
```

Syntax Rules

1) Specification of <Part 9 yes> implies that conformance to ISO/IEC 9075-9 is claimed.

2) Specification of <Part 9 datalink yes> implies that conformance to Feature M001, “Datalinks”, is claimed.

3) Specification of <Part 9 datalink no> implies that conformance to Feature M001, “Datalinks”, is not claimed.

4) Specification of <Part 9 wrapper yes> implies that conformance to Feature M002, “Foreign-data wrappers”, is claimed.

5) Specification of <Part 9 wrapper no> implies that conformance to Feature M002, “Foreign-data wrappers”, is not claimed.

4 Concepts

4.1 Data types

Insert after 7th paragraph SQL defines a predefined data type named by the following <key word>: DATALINK.

Insert after 8th paragraph For reference purposes, the data type DATALINK is referred to as a (or the) *datalink type*.

4.2 Foreign servers

A *foreign server* is an agency, external to the SQL-environment but known to the SQL-server, that manages external data. Such external data is manifested as SQL-data to the SQL-client by use of a mechanism called a *foreign-data wrapper* (see Subclause 4.3, "Foreign-data wrappers").

A foreign server is a catalog element, identified by a foreign server name and defined by invoking a <foreign server definition>. Invocation of a <foreign server definition> results in the creation of a foreign server descriptor in a catalog. A foreign server descriptor consists of:

— A foreign server name, identifying the foreign server being defined.

— The <authorization identifier> of the owner of the foreign server.

— The name of the foreign-data wrapper used to access the foreign server being defined.

— A generic options descriptor.

— Optionally, the foreign server type.

— Optionally, the foreign server version.

• 3 list elements deleted.

• 1 paragraph deleted.

The possible values of server type and server version, and their meanings, are implementation-defined.

• 1 paragraph deleted.

The foreign server object is said to be *owned by* or to have been *created by* the <authorization identifier> of the foreign server.

A foreign server descriptor can be modified by an <alter foreign server statement> and destroyed by a <drop foreign server statement>.

A foreign server can be an *SQL-aware foreign server* or a *non-SQL-aware foreign server*. An SQL-aware foreign server is a foreign server that has some ability to process SQL language (although not necessarily as a conforming SQL-server). A non-SQL-aware foreign server is a foreign server that

has no ability to process SQL language. If the foreign-data wrapper associated with a non-SQL-aware foreign server provide some (limited or conforming) ability to process SQL language, then the effect is that the foreign server can be treated as though it is an SQL-aware foreign server.

NOTE 2 – Some SQL-aware foreign servers may be, in fact, SQL-servers. However, because they are not in the same SQL-environment as the SQL-server responding to an SQL-client, they are managed only through foreign-data wrappers and are treated as foreign servers. Such foreign servers may concurrently respond to SQL-clients of their own; this does not change the relationships specified in this part of ISO/IEC 9075.

4.3 Foreign-data wrappers

A foreign-data wrapper is the mechanism by which the SQL-server accesses external data managed by zero or more foreign servers. A foreign-data wrapper is made up of foreign-data wrapper routines, a set of routines written in a standard programming language. Foreign-data wrapper routines are used to access every foreign server whose descriptor includes the name of that foreign-data wrapper. When a foreign-data wrapper is created, the name of a library that provides entry points for these routines is supplied. A foreign-data wrapper may be associated with multiple foreign servers.

A foreign-data wrapper is a catalog element, identified by a foreign-data wrapper name and defined by invoking a <foreign-data wrapper definition>. A <foreign-data wrapper definition> specifies the foreign-data wrapper name, a library name that identifies a list of entry points of the foreign-data wrapper routines, and the name of the language in which the foreign-data wrapper routines are written.

Invocation of a <foreign-data wrapper definition> results in the creation of a foreign-data wrapper descriptor in a catalog. A *foreign-data wrapper descriptor* consists of:

— A foreign-data wrapper name identifying the foreign-data wrapper being described.

— The <authorization identifier> of the owner of the foreign server.

— Name of the language in which the foreign-data wrapper routines are written.

— A generic options descriptor.

— Optionally, a library name.

• 1 list element deleted.

A foreign-data wrapper descriptor can be modified by an <alter foreign-data wrapper statement> and destroyed by a <drop foreign-data wrapper statement>.

• 1 paragraph deleted.

4.4 User mappings

A user mapping is an SQL-environment element, pairing an authorization identifier *U* with a foreign server *FS*. It defines how to map *U* to an equivalent concept known to *FS* when a foreign table whose source is *FS* is to be accessed during an SQL-session with current authorization identifier *U*. The mapping is specified by implementation-defined generic options.

A user mapping is defined by invoking an <user mapping definition>. Invocation of an <user mapping definition> results in the creation of a user mapping descriptor. A user mapping descriptor consists of:

— An authorization identifier.

— A foreign server name, identifying a foreign server descriptor.

— A generic options descriptor.

A user mapping descriptor can be modified by an <alter user mapping statement> and destroyed by a <drop user mapping statement>.

• 1 paragraph deleted.

4.5 Generic options

Several of the constructs used in connection with external data support the specification of *generic options*. A generic option is an option name paired with an optional option value. Both the option name and the option value of a generic option are implementation-defined. A set of generic options is described by a generic options descriptor. A generic options descriptor is included in the descriptor of the object to which it pertains.

A *generic options descriptor* is either an empty list or a list consisting of one or more option names, each option name being paired with at most one option value.

4.6 Capabilities and options information

The SQL-server needs information from the foreign-data wrapper about the capabilities of the foreign-data wrapper itself, about the foreign server accessed through the foreign-data wrapper, and about certain schema elements (foreign tables and their columns, user mappings) managed by the foreign server. The SQL-server also needs information about options supported by the foreign-data wrapper, the foreign server, and certain schema elements. The SQL-server invokes the GetOption( ) routine to request the capabilities and other information from a foreign-data wrapper.

The specific capabilities and other information of a foreign-data wrapper, a foreign server, or any schema element managed by a foreign server that are reported to the SQL-server in response to an invocation of GetOption are partly specified in this part of ISO/IEC 9075 and partly implementation-defined. In general, each capability or other piece of information that is reported corresponds to a generic option associated with the object being queried by the invocation.

The capabilities and other information is returned in a buffer whose contents comprise an XML document. The format of the XML document is TO BE DETERMINED.

• 1 Subclause deleted.

4.7 Datalinks

A datalink is a value of the DATALINK data type. A datalink represents some external data source (*i.e.*, one that is not part of the SQL-environment). The external data source is assumed to be accessible through some external data manager. A datalink conceptually consists of the following parts:

— Link Type: The type of linkage to an external file. The value is 'URL'.

— File Reference: A character string forming a reference to an external file, its format being determined by the Link Type.

— Comment: An arbitrary character string.

These conceptual parts are accessible by invoking built-in scalar functions defined in this part of ISO/IEC 9075.

The purpose of datalinks is to provide a mechanism to synchonize the integrity control, recovery, and access control of the external data sources and the SQL-data associated with them. This part of ISO/IEC 9075 standardizes the way that an SQL-server is made aware of datalink values and how applications retrieve information about the external data sources identified by datalink values. The mechanisms that enable integrity control, recovery, and access control for the external data sources represented by the datalink values are implementation-dependent. These mechanisms are collectively called the *datalinker*.

An external data source is "linked" to the SQL-environment whenever execution of an SQL-data change statement causes a value that references that external data source to appear in some datalink column whose descriptor includes the link control FILE LINK CONTROL. Execution of an SQL data change statement that causes a value to appear in a datalink column defined with the link control NO LINK CONTROL does not cause any external data source to be linked to the SQL-environment. A linked external data source cannot be renamed or deleted by any agency outside of the SQL-environment. One datalink value always references just one external data source. An external data source is "unlinked" from the SQL-environment whenever execution an SQL-data change statement causes a datalink that references that external data source to be removed from some datalink column whose descriptor includes the link control FILE LINK CONTROL. The actions that occur when a datalink is removed from a column depend on the link control options that are specified in the column descriptor of that column. The external data source may be deleted, or the datalinker can return control of the external data source to the external data manager.

With the function provided by datalinks and the datalinker, it is possible to specify that access to the external data sources should be controlled by the SQL-server rather than by the external data manager. When access to the external data sources is controlled by an SQL-server, any request to access an external data source must first retrieve the datalink value from a table and then obtain a charater string with which to reference the external data source, using one of the built-in functions provided for that purpose. This character string is constructed by combining the File Reference of a datalink value with an encrypted value called an *access token*. The generation of the access token and the method of combining it with the File Reference is implementation-dependent. When the application uses the returned character string value to access an external data source, the datalinker checks to see if the access token is *valid*. If it is valid, then the application is allowed to access the external data source pointed to by the File Reference. Every attempt by an application to access, without a valid access token, a linked external data source is unsuccessful. The time at which a valid access token ceases to be valid is implementation-defined.

A datalink data type is described by a *datalink data type descriptor*. A datalink data type descriptor is merely a data type descriptor with data type name DATALINK and no data type specific components.

The implementation-defined *maximum datalink length* determines the amount of space, in octets, that is allocated for:

— A host variable of data type DATALINK.

— An argument of declared type DATALINK to an invocation of an external routine.

— The value returned by an invocation of an external function whose result type is DATALINK.

The maximum datalink length constrains the values of expressions whose declared type is DATALINK such that every such value can be assigned to a host variable, substituted for a parameter to an external routine, or returned by an invocation of an external function.

4.7.1 Operations involving datalinks

4.7.1.1 Operators that operate on datalinks

<datalink comment expression> returns the Comment of a given datalink.

<link type expression> returns the Link Type of a given datalink.

<url complete expression> returns the complete URL of a given datalink whose Link Type is 'URL'.

<url path expression> returns the file path, including any access token, of the URL of a given datalink whose Link Type is 'URL'.

<url path only expression> returns the file path, excluding any access token, of the URL of a given datalink whose Link Type is 'URL'.

<url scheme expression> returns the scheme of the URL of a given datalink whose Link Type is 'URL'.

<url server expression> returns the file server of the URL of a given datalink whose Link Type is 'URL'.

4.7.1.2 Other operators involving datalinks

<datalink value constructor>, given a Link Type, File Reference, and Comment, returns the corresponding datalink.

4.8 Type conversion and mixing of data types

New paragraph A datalink is comparable only with values of data type DATALINK and assignable only to sites of data type DATALINK.

4.9 Columns, fields, and attributes

Insert following 8th paragraph The column descriptor of a column of the datalink type additionally includes:

— The link control (NO LINK CONTROL or FILE LINK CONTROL).

— The integrity control option (ALL, SELECTIVE, or NONE).

— The read permission option (FS or DB).

— The write permission option (FS or BLOCKED).

— The recovery option (NO or YES).

— The unlink option (RESTORE, DELETE, or NONE).

New paragraph The meanings of the various link control options are:

— NO LINK CONTROL: Although every file path must conform to the syntax for such identifiers as specified by the external file server, it is permitted for there to be no external data source referenced by that file path. This option implies that the integrity control option is NONE, the read permission option is FS, the write permission option is FS, the recovery option is NO and the unlink option is NONE, and no explicit syntax to specify these options is permitted.

— FILE LINK CONTROL: Every file path must reference an existing external data source. Further file control depends on the link control options.

— INTEGRITY ALL: External data sources referenced by file paths cannot be deleted or renamed, except possibly through the use of operations on the column in question, invoked as part of some SQL-session.

— INTEGRITY SELECTIVE: External data sources referenced by file paths can be deleted or renamed using operators provided by the file server, unless a datalinker is installed in connection with the file server.

— INTEGRITY NONE: External data sources referenced by file paths can only be deleted or renamed using operators provided by the file server. This option is not available if FILE LINK CONTROL is specified.

— READ PERMISSION FS: Permission to read external data sources referenced by datalinks is determined by the file server.

— READ PERMISSION DB: Permission to read external data sourcees referenced by datalinks is determined by the SQL-implementation.

— WRITE PERMISSION FS: Permission to write external data sources referenced by datalinks is determined by the file server.

— WRITE PERMISSION BLOCKED: Write access to external data sources referenced by datalinks is not available. Updates can, however, arise indirectly through the use of some implementation-defined mechanism.

— RECOVERY YES: Enables *point in time recovery* of external data sources referenced by datalinks.

NOTE 3 – "point in time recovery" is an implementation-defined mechanism that provides for recovery that is coördinated between the SQL-server and the files of external file systems referenced by datalinks.

— RECOVERY NO: Point in time recovery of external data sources referenced by datalinks is disabled.

— ON UNLINK RESTORE: When an external data source referenced by a datalink is unlinked, the external file server attempts to reinstate the ownership and permissions that existed when that object was linked.

— ON UNLINK DELETE: An external data source referenced by a datalink is deleted when it is unlinked.

— ON UNLINK NONE: When an external data source referenced by a datalink is unlinked, there is no change in the ownership and permissions occasioned by that unlinking.

New paragraph Table 2, "Valid datalink file control options", specifies what combinations of datalink file control options are allowed.

New paragraph The default value of a column whose declared type is DATALINK is the null value. Datalinks are subject to certain restrictions. As a consequence of these restrictions, datalinks and columns of declared type DATALINK cannot be referenced in (among other places):

— <general set function>.

— <group by clause>.

— <order by clause>.

— <unique constraint definition>.

— <referential constraint definition>.

— <select list> of a <query specification> that has a <set quantifier> of DISTINCT.

— UNION, INTERSECT, and EXCEPT.

— Columns used for matching when forming a <joined table>.

Table 2—Valid datalink file control options

Integrity	Read permission	Write permission	Recovery	Unlink
ALL	FS	FS	NO	NONE
ALL	FS	BLOCKED	NO	RESTORE
ALL	FS	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	NO	RESTORE
ALL	DB	BLOCKED	NO	DELETE

Table 2—Valid datalink file control options (Cont.)

Integrity	Read permission	Write permission	Recovery	Unlink
ALL	DB	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	YES	DELETE
SELECTIVE	FS	FS	NO	NONE

4.10 Tables

Replaces 3rd paragraph A table is either a base table, a derived table, or a foreign table. A base table is either a persistent base table, a global temporary table, a created local temporary table, or a declared local temporary table.

****Editor's Note****

We need to decide if a foreign table is a new variety of table, a new subvariety of base table, or perhaps something else (Steve Cannan has suggested that it is a kind of derived table). Note that currently Clause 3, "Definitions, notations, and conventions", and Clause 4, "Concepts", disagree on this point.

See Possible Problem **MED-022**.

Replaces 10th paragraph A table is described by a table descriptor. A table descriptor is either a base table descriptor, a view descriptor, a derived table descriptor (for a derived table that is not a view), or a foreign table descriptor.

Insert this paragraph The data constituting a foreign table is not part of the SQL-environment. Instead, its rows are supplied when needed by some foreign server, known as the *source* of the foreign table. The mechanism by which these rows are supplied is provided by a *foreign-data wrapper* (see Subclause 4.3, "Foreign-data wrappers"). A foreign table descriptor describes a foreign table. In addition to the components of every table descriptor, a foreign table descriptor includes:

— The name of the foreign table.

— A foreign server name, identifying the descriptor of the foreign server that is the source of the foreign table.

— A generic options descriptor.

• 1 paragraph deleted.

4.11 SQL-statements

4.11.1 SQL-statements classified by function

New paragraph The following are additional SQL-schema statements:

• 1 list element deleted.

— <foreign table definition>

— <alter foreign table statement>

— <drop foreign table statement>

— <foreign server definition>

— <alter foreign server statement>

— <drop foreign server statement>

— <foreign-data wrapper definition>

— <alter foreign-data wrapper statement>

— <drop foreign-data wrapper statement>

— <user mapping definition>

— <alter user mapping statement>

— <drop user mapping statement>

4.12 SQL-sessions

Insert this paragraph At any time during an SQL-session, the SQL-server may obtain a WrapperEnvHandle for a foreign-data wrapper and an FSConnectionHandle for a foreign server.

Insert this paragraph The SQL-session context also comprises:

— Zero or more {foreign-data wrapper name : WrapperEnvHandle} pairs.

— Zero or more {foreign server name : FSConnectionHandle} pairs.

4.13 Privileges

Replace the 8th paragraph A privilege descriptor with an <action> of USAGE is called a *usage privilege descriptor* and identifies the existence of a privilege on the domain, user-defined type, character set, collation, foreign-data wrapper, foreign server, or translation identified by the privilege descriptor.

4.14 Foreign-data wrapper interface

A foreign-data wrapper interface comprises routines that:

— Allocate and deallocate resources.

— Control connections to foreign servers.

— Receive information and/or data from the SQL-server about the SQL statement to be executed at the foreign server.

— Send information and/or data from the foreign server to the SQL-server about the SQL statement that the foreign server is willing to execute.

— Initiate and terminate the execution the execution of SQL statements by the foreign server.

4.14.1 Handles

A *handle* is a value of INTEGER data type that uniquely identifies an encapsulated information. A set of `Getxxx`routines associated with a handle provide access to individual components of the encapsulated information. Although the declared type for a handle is INTEGER, its value has no meaning in any other context and should not be used as a numeric operand or modified in any way.

The following are the handles specified in the foreign-data wrapper interface:

— **WrapperEnv handle**: This handle is allocated by a foreign-data wrapper to maintain any necessary information during the interaction with an SQL-server. It is allocated via a call to the AllocWrapperEnv( ) routine. This handle must be allocated before any requests are made to a foreign-data wrapper.

— **Server handle**: This handle is allocated by an SQL-server to describe key attributes of a foreign server. A foreign-data wrapper uses this handle to obtain information about the foreign server it needs to connect. Routines associated with a server handle allow information to be obtained about such things as the server name, server type, server version, *etc.* This handle is allocated implicitly.

— **FSConnection handle**: This handle is allocated by a foreign-data wrapper to maintain information associated with a connection to a foreign server. It is allocated via a call to the ConnectForiegnServer( ) routine. This handle must be allocated before any SQL statements are sent to a foreign-data wrapper.

— **Request handle**: This handle is allocated by an SQL-server to describe an SQL-statement that is to be executed by a foreign server. A request handle may describe a simple statement, such as `SELECT * FROM T`, or it may describe a complex statement that includes predicates, joins, ordering, *etc.* Routines associated with a request handle allow foreign-data wrapper to retrieve the names of foreign tables referenced in the from clause, the names of columns references in the select list, *etc.* This handle is allocated implicitly.

— **Table Reference handle**: This handle is allocated by an SQL-server to describe a <table reference> contained in the <from clause> of a <query specification>. This handle is allocated implicitly.

— **Value Expression handle**: This handle is allocated by an SQL-server to describe a <value expression> contained in the <select list> of a <query specification>. This handle is allocated implicitly.

— **Reply handle**: This handle is allocated by a foreign-data wrapper to describe the subset of SQL-statements sent by an SQL-server it is capable of executing. This handle is allocated via a call to the InitForeignRequest( ) routine.

— **Execution handle**: This handle is allocated by a foreign-data wrapper to describe the information it needs to process the SQL-statement described by a corresponding reply handle and the information associated with the data resulting from the processing of the SQL-statement. This handle is also allocated via a call to the InitForeignRequest( ) routine.

• 2 list elements deleted.

— **Wrapper handle**: This handle is allocated implicitly by an SQL-server to describe the information about a foreign-data wrapper.

— **User handle**: This handle is allocated implicitly by an SQL-server to describe information about the user on whose behalf the connection to the foreign server is being made.

— **Descriptor handle**: This handle is allocated by either an SQL-server or a foreign-data wrapper to maintain information about an MED descriptor area.

4.14.2 Foreign-data wrapper interface routines

There are four main classes of foreign-data wrapper interface routines: handle routines, initialization routines, access routines, and termination routines.

4.14.2.1 Handle routines

— **GetServerName**: This routine returns the name of a foreign server given a server handle.

— **GetServerType**: This routine returns the type of a foreign server given a server handle.

— **GetServerVersion**: This routine returns the version of a foreign server given a server handle.

— **GetNumberOfServerOptions**: This routine returns the number of generic options associated with a foreign server given a server handle.

— **GetServerOption**: This routine returns the generic option name and its value given a server handle and the position of the option in the options list.

— **GetServerOptionValueByName**: This routine returns the generic option value given a server handle and the name of the option.

— **GetNumberOfTableReferenceElements**: This routine returns the number of <table reference>s in the <from clause> of a query given a request handle.

— **GetTableReferenceElement**: This routine returns the table reference handle of a <table reference> in the <from clause> of a query given a request handle and the position of the <table reference> in the <from clause>.

— **GetTableReferenceElementType**: This routine returns the "type" of a <table reference> given the table reference handle. The only possible return value is TABLE_NAME.

— **GetTableReferenceTableName**: This routine returns the table name given a table reference handle.

— **GetNumberOfSelectListElements**: This routine returns the number of <value expression>s in the <select list> of a <query specification> given a request handle.

— **GetSelectListElement**: This routine returns the value expression handle of a <value expression> in the <select list> of a <query specification> given a request handle and the position of the <value expression> in the <select list>.

— **GetSelectListElementType**: This routine returns the "type" of a <value expression> given the value expression handle. The only possible return value is COLUMN_NAME.

— **GetValueExpressionColumnName**: This routine returns the name of the column, given a value expression handle.

— **GetNumberOfReplyTableReferences**: This routine returns the number of table references from the original request that the foreign-data wrapper is capable of accessing, given a reply handle.

— **GetReplyTableRefernce**: This routine returns the number of the table reference in the original request that the foreign-data wrapper is capable of accessing, given a reply handle and a number that ranges from 1 (one) to the value returned by the GetNumberOfReplyTableReferences( ) routine.

— **GetNumberOfReplySelectListElements**: This routine returns the number of select list elements from the original request that the foreign-data wrapper is capable of accessing, given a reply handle.

— **GetReplySelectListElement**: This routine returns the number of the select list element in the original request that the foreign-data wrapper is capable of accessing, given a reply handle and a number that ranges from 1 (one) to the value returned by the GetNumberOfReplySelectListElements( ) routine.

• 5 list elements deleted.

— **GetAuthorizationId**: This routine returns the authorization identifier associated with a user mapping, given a user handle.

— **GetForeignTableColumn**: This routine returns the name and data type of a column, given a table reference handle and the position of the column in the column list.

— **GetForeignTableColumnOption**: This routine returns the generic option name and its value, given a table reference handle, column name and the position of the option in the options list.

— **GetForeignTableColumnOptionValueByName**: This routine returns the generic option value, given a table reference handle, a column name and the name of the option.

— **GetForeignTableColumnType**: This routine returns the data type of a column, given a table reference handle and the name of the column.

— **GetForeignTableOption**: This routine returns the generic option name and its value, given a table reference handle and the position of the option in the options list.

— **GetForeignTableOptionValueByName**: This routine returns the generic option value, given a table reference handle and the name of the option.

— **GetForeignTableServerName**: This routine returns the name of the foreign server associated with a foreign table, given a table reference handle.

— **GetNumberOfForeignTableColumnOptions**: This routine returns the number of generic options associated with a column of a foreign table, given a table reference handle and a column name.

— **GetNumberOfForeignTableColumns**: This routine returns the number of columns associated with a foreign table, given a table reference handle.

— **GetNumberOfForeignTableOptions**: This routine returns the number of generic options associated with a foreign table, given a table reference handle.

— **GetNumberOfUserOptions**: This routine returns the number of generic options associated with a user mapping, given a user handle.

— **GetNumberOfWrapperOptions**: This routine returns the number of generic options associated with a foreign-data wrapper, given a wrapper handle.

— **GetUserOption**: This routine returns the generic option name and its value, given a user handle and the position of the option in the options list.

— **GetUserOptionValueByName**: This routine returns the generic option value, given a user handle and the name of the option.

— **GetWrapperLibraryName**: This routine returns the name of the library associated with a foreign-data wrapper, given a wrapper handle.

— **GetWrapperName**: This routine returns the name of a foreign-data wrapper, given a wrapper handle.

— **GetWrapperOption**: This routine returns the generic option name and its value, given a wrapper handle and the position of the option in the options list.

— **GetWrapperOptionValueByName**: This routine returns the generic option value, given a wrapper handle and the name of the option.

— **GetDescriptor**: This routine, given a descriptor handle and the identification of a descriptor area field, retrieves the value of the specified field from a descriptor area.

— **SetDescriptor**: This routine, given a descriptor handle, the identification of a descriptor area field, and a new value to be assigned to that field, sets the value of the specified field of a descriptor area.

— **GetSPDHandle**: This routine returns the SPDHandle given an ExecutionHandle.

— **GetSRDHandle**: This routine returns the SRDHandle given an ExecutionHandle.

— **GetTRDHandle**: This routine returns the TRDHandle given an TableReferenceHandle.

— **GetWPDHandle**: This routine returns the WPDHandle given an ExecutionHandle.

— **GetWRDHandle**: This routine returns the WRDHandle given an ExecutionHandle.

4.14.2.2 Initialization routines

— **AllocDescriptor**: This routine is used by the SQL-server to request that the foreign-data wrapper allocate a descriptor area for use in exchanging information about values required to execute an SQL-statement or values expected to be returned from an execution of an SQL-statement.

— **AllocWrapperEnv**: This routine allows a foreign-data wrapper to perform any initialization steps and allocate and initialize any necessary global data structures. It has a single input parameter, a WrapperHandle, that describes the information about the foreign-data wrapper maintained by the SQL-server, and a single output parameter, a WrapperEnv handle, which is a handle to the foreign-data wrapper's newly initialized global data structures.

— **ConnectForeignServer**: This routine is used by the SQL-server to request access to a foreign server, and allows the foreign-data wrapper associated with that foreign server to establish a connection (if necessary) and set up any required state information. ConnectForeignServer has three input parameters: a previously allocated WrapperEnv handle, a server handle that describes the foreign server for which the SQL-server is requesting a connection, and a UserHandle that describes the user mapping maintained by the SQL-server. The routine has one output parameter, the newly allocated FSConnection handle for the foreign server.

— **GetOptions**: This routine is used by the SQL-server to request that the foreign-data wrapper return information about the capabilities and other aspects of the foreign-data wrapper, the foreign server, some foreign table at the foreign server, or some foreign column of some foreign table at the foreign server.

— **InitForeignRequest**: This routine is used by the SQL-server to cause an SQL-statement—either one previously transmitted to the foreign-data wrapper through the use of TransmitForeignRequest, or one submitted through a query handle parameter of InitForeignRequest—to be executed. The routine has two input parameters: a previously allocated FSConnection handle, and a request handle that describes the data retrieval request. The routine has two output parameters: a reply handle that describes how much of the request the foreign-data wrapper is willing to handle, and an execution handle to the state information the foreign-data wrapper needs to process the request and the data rows that will be returned.

4.14.2.3 Access routines

— **Open**: This routine is used by the SQL-server to allow the foreign-data wrapper to allocate any resources necessary to perform the operations represented by the ExecutionHandle (and described by a ReplyHandle previously returned by the InitForeignRequest routine). The routine has one input parameter: a previously allocated ExecutionHandle.

— **Iterate**: This routine is used by the SQL-server to iteratively retrieve data from a foreign-data wrapper. The routine has one input parameter, a previously allocated ExecutionHandle. As a result of this call, the foreign-data wrapper will associate the next set of rows with the ExecutionHandle. The SQL-server may invoke this routine until all data is returned.

— **ReOpen**: This routine is used by the SQL-server to allow a foreign-data wrapper to re-initialize any resources necessary to re-execute the operations represented by the ExecutionHandle (and described by a ReplyHandle previously given in the InitForeignRequest routine). This routine allows a SQL-server to re-execute the operations associated with an ExecutionHandle multiple times, for example, if the work to be done by the foreign-data wrapper represents the inner node of a join being processed by the SQL-server. The routine has one input parameter: a previously allocated ExecutionHandle.

— **Close**: This routine is used by the SQL-server to allow a foreign-data wrapper to free any resources that had been allocated to perform the operations represented by the ExecutionHandle. The SQL-server invokes this routine after it is done processing an SQL-statement that initiated the communication with the foreign-data wrapper. The routine has one input parameter: a previously allocated ExecutionHandle.

— **RetrieveStatistics**: : This routine is used by the SQL-server to request statistics, if any, related to the SQL-statement previously sent to the foreign-data wrapper. Such statistics are entirely implementation-defined in nature.

— **TransmitForeignRequest**: : This routine is used by the SQL-server to transmit an SQL-statement to the foreign-data wrapper. The foreign-data wrapper, possibly in cooperation with the foreign server with which it is associated, analyzes the transmitted SQL-statement and returns information about that SQL-statement to the SQL-server. This information includes: Whether the SQL-statement is a SELECT statement, some SQL-statement other than a SELECT statement, or contains a syntax error; whether the SQL-statement requires one or more values in order to be executed; and whether the SQL-statement returns one or more values upon execution.

4.14.2.4 Termination routines

• 1 list element deleted.

— **FreeReplyHandle**: This routine is used by the SQL-server to allow the foreign-data wrapper to free the resources associated with a ReplyHandle after the SQL-server has determined that it no longer needs the information encapsulated by the ReplyHandle. This routine has one input parameter, the previously allocated ReplyHandle.

— **FreeExecutionHandle**: This routine is used by the SQL-server to allow the foreign-data wrapper to free the resources associated with an ExecutionHandle after the SQL-server has determined that it no longer needs the information encapsulated by the ExecutionHandle. This routine has one input parameter, a previously allocated ExecutionHandle.

— **FreeFSConnection**: This routine is used by the SQL-server to terminate a connection to a foreign server. It allows the foreign-data wrapper to disconnect from the foreign server and to free any resources associated with the connection, such as the FSConnection handle. It has one input parameters: a previously allocated FSConnection handle. The routine has no output parameters.

— **FreeWrapperEnv**: This routine is used by the SQL-server to terminate communication with a foreign-data wrapper. It allows the foreign-data wrapper to free any global resources it had allocated, such as the WrapperEnv handle. The routine has one input parameter, the previously allocated WrapperEnv handle. The routine has no output parameters.

— **FreeDescriptor**: This routine is used by the SQL-server to request that the foreign-data wrapper deallocate a descriptor area and to free the memory and other resources used by that descriptor.

4.14.2.5 Sequence of actions during the execution of requests involving foreign tables

Table 3—Sequence of actions during foreign server request executions

Step	SQL-server	Flow	Foreign-data wrapper
1	Receives a query from SQL-client that involves data from a foreign table in a <table reference>, possibly in "pass-through" mode. Determines relationship of foreign table → foreign server → foreign-data wrapper. For SQL-aware foreign servers, decomposes the SQL-statement into separate SQL-statements that each involve no more than a single foreign server or local SQL-server. (The remainder of this table considers only the behavior of a single SQL-statement (possibly a "sub-SQL-statement") involving a single foreign server.)		
2	Creates a **WrapperHandle** and associates information about the foreign-data wrapper with that handle.		
3	Invokes **AllocWrapperEnv (WrapperHandle, &WrapperEnvHandle)** to initialize the foreign data wrapper.	⇒	
4		⇐	Invokes the **Get... (WrapperHandle, ...)** routines in the SQL-server to retrieve information about the foreign-data wrapper known to the SQL-server.
5	Executes the **Get... (WrapperHandle, ...)** routines as requested from the foreign-data wrapper.	⇒	
6		⇐	Allocates global data structures associated with the wrapper and associates them with the **WrapperEnvHandle** and performs any initialization required.
7	Frees the **WrapperHandle**.		
8	Creates a **ServerHandle** and associates information about the foreign server with that handle.		
9	Creates a **UserHandle** and associates information about the current user with that handle.		

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
10	Invokes **ConnectForeignServer (WrapperEnvHandle, ServerHandle, UserHandle, &FSConnectionHandle)** in the foreign-data wrapper to establish a connection to the foreign server that contains some of the data required to process the SQL-server client's query.	⇒	
11		⇐	Invokes the **Get... (ServerHandle, ...)** routines in the SQL-Server to retrieve all information about the foreign server known to the SQL-server.
12	Executes the **Get... (ServerHandle, ...)** routines as requested from the foreign data wrapper.	⇒	
13		⇐	Invokes the **Get... (UserHandle, ...)** routines in the SQL-Server to retrieve all information about the user known to the SQL-server.
14	Executes the **Get... (UserHandle, ...)** routines as requested from the foreign data wrapper.	⇒	
15		⇐	Allocates global data structures associated with the foreign server and associates them with a **FSConnectionHandle**, establishes a connection to the foreign server, and performs any initialization required.
16	Frees the **ServerHandle**.		
17	Frees the **UserHandle**.		
18	Creates a **RequestHandle** and associates with it the information about the part of query that could be handled by the foreign-data wrapper. (For SQL-aware foreign servers, this will be either the sub-SQL-statement or the "pass-through" SQL-statement.)		
19	(For non-SQL-aware foreign servers only:) Creates as many **TableReferenceHandles** as are needed and associates with each of them the information about a particular <table reference>. (Note: This step may be performed any time before Step FIX 22 FIX.)		

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
20	(For non-SQL-aware foreign servers only:) Create a Table Reference Descriptor (TRD) for the foreign table in the <table reference>, set all the fields with details about each of the columns and associate its handle to the corresponding TableReferenceHandle.		
21	(For non-SQL-aware foreign servers only:) Creates as many **ValueExpressionHandles** as are needed and associates with each of them the information about a particular <value expression> in the <select list>. (Note: This step may be performed any time before Step FIX 22 FIX.)		
22	(For "SQL-aware" foreign servers in pass-through mode only:) Invoke **TransmitForeignRequest (FSConnectionHandle, RequestHandle, &ReplyHandle)** to allow the foreign-data wrapper and the foreign server to analyze the SQL-statement and return information about the type of statement (SELECT, non-SELECT, non-Data, or Invalie), the number of input value required for its execution, and (if SELECT) the number of select-list columns to be returned.	⇒	
23		⇐	(For "SQL-aware" foreign servers in pass-through mode only:) Executed the **TransmitForeignRequest** function, returning information about the type of SQL-statement, the number of input values and the number of select-list columns.

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
24	(For "SQL-aware" foreign servers only:) If the number of input values is greater than zero, invoke **AllocDescriptor (FSConnectionHandle, RequestHandle, NumberOfInputValues, &DescriptorHandle)**. If the SQL-statement is a "pass-through" SQL-statement, then the information about the number and data types of any input values and the number and data types of any output values are known from the **TransmitForeignRequest** invocation. If the SQL-statement is a sub-SQL-statement, then the local SQL-server already knows this information, since it constructed (or at least analyzed) the statement to be executed; thus, no interaction is required with the foreign-data wrapper to determine the number of input values or their data types. Instead, the descriptor area is used only to provide the input values themselves.	⇒	
25			(For "SQL-aware" foreign servers only:) Allocates a descriptor area of the specified size.
26	(For "SQL-aware" foreign servers in pass-through mode only:) If the number of input values is greater than zero, then invoke **DescribeInput (FSConnectionHandle, RequestHandle, DescriptorHandle)**.	⇒	
27			(For "SQL-aware" foreign servers in pass-through mode only:) Populate the descriptor area with a description of the input values required for the SQL-statement.
28	(For "SQL-aware" foreign servers in pass-through mode only:) If the number of select-list columns is greater than zero, then invoke **AllocDescriptor (FSConnectionHandle, RequestHandle, NumberOfSelectListColumns, &DescriptorHandle)**	⇒	

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
29			(For "SQL-aware" foreign servers in pass-through mode only:) Allocates a descriptor area of the specified size.
30	(For "SQL-aware" foreign servers in pass-through mode only:) If the number os select-list columns is greater than zero, then invoke **DescribeOutput (FSConnectionHandle, RequestHandle, DescriptorHandle)**	⇒	
31			(For "SQL-aware" foreign servers in pass-through mode only:) Populate the descriptor area with a description of the select-list columns of the SQL-statement.
32	(For "SQL-aware" foreign servers only:) If the number of input values is greater than zero, then for each input value, invoke **SetDescriptor (FSConnectionHandle, DescriptorHandle, DetailNumber, Value)**.	⇒	
33			(For "SQL-aware" foreign servers only:) Store the value provided into the specified detail area of the descriptor area.
34	Invokes **InitForeignRequest (FSConnectionHandle, RequestHandle, &ReplyHandle, &ExecutionHandle)** in the foreign-data wrapper to find out how much of the request the foreign server can actually process. (For "SQL-aware" foreign servers only:) If the statement is a SELECT statement, then this routine is complete. If the statement is valid, but is something other than a SELECT statement, then the statement actually gets executed here!	⇒	
35		⇐	(For non-SQL-aware foreign servers only:) Invokes the **Get...(RequestHandle, ...)**, **Get...(TableReferenceHandle, ...)**, and **Get...(ValueExpressionHandle, ...)** routines in the SQL-server to examine the SQL-server's request and to determine how much of the request the foreign data wrapper can handle.

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
36			(For non-SQL-aware foreign servers only:) Invoke **GetTRDHandle (TableReferenceHandle)** to get the TRD and then invoke **GetDescriptor (TRD)** to determine all the details of the columns of the foreign table referenced in <table reference>.
37	(For non-SQL-aware foreign servers only:) Executes the **Get...(RequestHandle, ...)**, **Get...(TableReferenceHandle, ...)**, and **Get...(ValueExpressionHandle, ...)** routines as requested by the foreign data wrapper.	⇒	
38			(For non-SQL-aware foreign servers only:) Creates a **ReplyHandle** and associates with it the information about the part of query that could actually be handled by the foreign-data wrapper.
39		⇐	Creates an **ExecutionHandle** and associates with it the information about the actual execution plan. (For "SQL-aware" foreign servers *not* in pass-through mode only:) If the statement was a valid statement other than a SELECT statement, then the statement gets executed immediately and the status of that execution is returned along with the **ExecutionHandle**.
40		⇐	(For non-SQL-aware foreign servers only:) Create two implicit descriptors (WRD and SRD) to describe the columns of the result table and associate both of them to the **ExecutionHandle**. Initialize both the descriptors with default values wherever applicable. Set the fields in the WRD to correspond to the result columns associated with the **ExecutionHandle**.

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
41			(For non-SQL-aware foreign servers only:) Create two implicit descriptors, WPD and SPD, to describe <dynamic parameter specification>s. Initialize both the descriptors with default values wherever applicable. Set the fields in the WPD to correspond to the dynamic parameters associated with the **ExecutionHandle**.
42	Frees the **RequestHandle**.		
43	(For non-SQL-aware foreign servers only:) Frees each of the **TableReferenceHandle**s.		
44	(For non-SQL-aware foreign servers only:) Frees each of the **ValueExpressionHandle**s.		
45	(For non-SQL-aware foreign servers only:) Invokes the **Get...(ReplyHandle, ...)** routines in the foreign-data wrapper to incorporate the work that a wrapper can do into the execution plan for the SQL-server client's query.	⇒	
46		⇐	(For non-SQL-aware foreign servers only:) Executes the **Get...(ReplyHandle, ...)** routines as requested by the SQL-server.
47	(For non-SQL-aware foreign servers only:) Invokes **FreeReplyHandle (ReplyHandle)** in the foreign-data wrapper to indicate that the replyHandle is no longer required.	⇒	
48		⇐	(For non-SQL-aware foreign servers only:) Frees resources associated with **ReplyHandle**.
49	(For non-SQL-aware foreign servers only:) Invoke **GetWRDHandle (ExecutionHandle)** and **GetSRDHandle (ExecutionHandle)** to get the WRD and the SRD respectively.		

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
50	(For non-SQL-aware foreign servers only:) Invoke **GetDescriptor (WRD)** multiple times to get all the information associated with WRD.		
51	(For non-SQL-aware foreign servers only:) Invoke **SetDescriptor (SRD)** multiple times to populate appropriate fields in SRD.		
52	(For non-SQL-aware foreign servers only:) Invoke **GetWPDHandle (ExecutionHandle)** and **GetSPDHandle (ExecutionHandle)** to get the WPD and the SPD respectively.		
53	(For non-SQL-aware foreign servers only:) If there are any dynamic parameters present, invoke **GetDescriptor (WPD)** multiple times to obtain the information associated with WPD.		
54	(For non-SQL-aware foreign servers only:) If there are any dynamic parameters present, invoke **SetDescriptor (SPD)** multiple times to populate appropriate fields in SPD.		
55	(For SQL-aware foreign servers only, do this step if the statement is a SELECT statement:) Prepares to execute the plan for the SQL-server client's query. Invokes **Open (ExecutionHandle)** in the foreign-data wrapper to allow the foreign-data wrapper to prepare for query execution.	⇒	
56	(For SQL-aware foreign servers only, do this step if the statement is a SELECT statement:) Invokes **Iterate (ExecutionHandle)** in the foreign-data wrapper to retrieve the next set of rows.	⇒	
57		⇐	Performs the work needed to retrieve the next set of rows from the foreign server and associates it with the **ExecutionHandle**.

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
58	(For SQL-aware foreign servers only, do this step if the statement is a SELECT statement:) Invokes **Get...(ExecutionHandle)** routines in the foreign-data wrapper to retrieve the next set of rows from the foreign-data wrapper.	⇒	
59		⇐	Executes the **Get...(ExecutionHandle)** routines as requested by the SQL-Server.
60	(For SQL-aware foreign servers only, do this step if the statement is a SELECT statement:) Repeats steps **Iterate (ExecutionHandle)** ... **Get...(ExecutionHandle)** routines until all data is retrieved.		
61	(For SQL-aware foreign servers only, do this step if the statement is a SELECT statement:) Optional: If the work performed by the wrapper needs to be repeated, the SQL-server may choose to invoke **ReOpen (ExecutionHandle)** routine in the foreign-data wrapper to allow the wrapper to prepare to re-execute the query. (This is intended for use in cases where the foreign server is providing information in an inner query loop.)	⇒	
62		⇐	Optional: Performs the steps necessary to reuse the resources allocated in the **Open()** call in order to re-execute. In the worst case, it may need to redo everything done in the **Open()** call. In the average case, it may only need to reset counters, cursors, *etc*.
63	(For SQL-aware foreign servers only, do this step if the statement is a SELECT statement:) Completes the work necessary to answer the SQL-client's query. As a result, invokes **Close (ExecutionHandle)** routine in the foreign data wrapper.	⇒	
64			Executes the **Close (ExecutionHandle)** routine as requested by the SQL-server.
65	Invokes **FreeExecutionHandle (ExecutionHandle)** routine in the foreign data wrapper.	⇒	
66		⇐	Frees resources associated with **ExecutionHandle**.

Table 3—Sequence of actions during foreign server request executions (Cont.)

Step	SQL-server	Flow	Foreign-data wrapper
67	(For SQL-aware foreign servers only:) If a descriptor area was allocated for input values, then invoke **FreeDescriptor (FSConnectionHandle, DescriptorHandle)**.	⇒	
68		⇐	(For SQL-aware foreign servers only:) Execute FreeDescriptor to release resources associated with the input value descriptor area.
69	(For SQL-aware foreign servers only:) If a descriptor area was allocated for select-list columns, then invoke **FreeDescriptor (FSConnectionHandle, DescriptorHandle)**.	⇒	
70		⇐	(For SQL-aware foreign servers only:) Execute FreeDescriptor to release resources associated with the select-list column descriptor area.
71	Invokes **FreeFSConnection (FSConnectionHandle)** routine in the foreign-data wrapper.	⇒	
72		⇐	Frees resources associated with **FSConnectionHandle**.
73	Invokes **FreeWrapperEnv (WrapperEnvHandle)** routine in the foreign-data wrapper.	⇒	
74		⇐	Frees resources associated with **WrapperEnvHandle**.

****Editor's Note****
The preceding table is a merger of the corresponding tables in WG3:RTM-099R1/X3H2-99-398R1 and WG3:RTM-058R2/X3H2-99-314R4. However, the changes in descriptor treatment in WG3:RTM-099R1/X3H2-99-398R1 have not yet been reflected in the sections of the table describing the algorithm proposed by WG3:RTM-058R2/X3H2-99-314R4. That must be done before progression to FDIS. See Possible Problem **MED-036**.

4.14.3 Return codes

The execution of a foreign-data interface routine causes one or more conditions to be raised. The status of the execution is indicated by a code that is returned either as the result of a foreign-data wrapper routine that is a foreign-data wrapper function or as the value of the ReturnCode parameter of a foreign-data wrapper routine that is a foreign-data wrapper procedure. The values and meanings of the return codes are as follows. If more than one return code is possible, then the one appearing later in the list is the one returned.

— A value of 0 (zero) indicates **Success**. The foreign-data wrapper routine executed successfully.

— A value of 100 indicates **No data found**. The foreign-data wrapper routine executed successfully but a completion condition was raised: no data.

— A value of −1 indicates **Error**. The foreign-data wrapper routine did not execute successfully. An exception condition other than *foreign-data wrapper-specific condition — invalid handle* was raised.

— A value of −2 indicates **Invalid handle**. The foreign-data wrapper routine did not execute successfully because an exception condition was raised: *foreign-data wrapper-specific condition — invalid handle*.

If the foreign-data wrapper routine did not execute successfully, then the values of all output arguments are implementation-dependent unless explicitly defined by this part of ISO/IEC 9075.

In addition to providing the return code, for all SQL/MED routines other than GetDiagnostics(), the implementation records information about completion conditions and about exception conditions raised other than *foreign-data wrapper-specific condition — invalid handle* in the diagnostics area associated with the resource being utilized.

The resource being utilized by a routine is the resource identified by its input handle. In case of routines that have multiple input handles, the resource being utilized is deemed to be the one identified by the handle that comes first in the parameter list, with one exception: in the case of AllocWrapperEnv( ) routine, diagnostics are returned on the output parameter, WrapperEnvHandle, provided an allocated FDW-environment is successfully created; otherwise, no diagnostics are returned.

4.14.4 Diagnostics areas in SQL/MED

Each diagnostics area comprises header fields that contain general information relating to the routine that was executed and zero or more status records containing information about individual conditions that occurred during the execution of the MED routine. A condition that causes a status record to be generated is referred to as a status condition.

At the beginning of the execution of any MED routine other than GetDiagnostics( ), the diagnostics area for the resource being utilized is emptied. If the execution of such a routine does not result in the exception condition being raised: *foreign-data wrapper-specific condition — invalid handle*, then:

— Header information is generated in the diagnostics area.

— If the routine's return code indicates **Success**, then no status records are generated.

— If the routine's return code indicates **Success with information** or **Error**, then one or more status records are generated.

— If the routine's return code indicates **No data found**, then no status record is generated corresponding to SQLSTATE value '02000' but there may be status records generated corresponding to SQLSTATE value '02*nnn*', where '*nnn*' is an implementation-defined subclass value.

Status records in the diagnostics area are placed in an order that is implementation-dependent except that:

— For the purpose of choosing the first status record, status records corresponding to transaction rollback have precedence over status records corresponding to other exceptions, which in turn have precedence over status records corresponding to the completion condition *no data*, which in turn have precedence over status records corresponding to the completion condition *warning*.

— Apart from any status records corresponding to an implementation-specified *no data*, any status record corresponding to an implementation-specified condition that duplicates, in whole or in part, a condition defined in this part of ISO/IEC 9075 shall not be the first status record.

The GetDiagnostics( ) routine retrieves information from a diagnostics area. The application identifies which diagnostics area is to be accessed by providing the handle of the relevant resource as an input argument. The GetDiagnostics( ) routine returns a result code but do not modify the identified diagnostics area.

An SQL/MED diagnostics area comprises the fields specified in Table 4, "Fields used in SQL/MED diagnostics areas".

Table 4—Fields used in SQL/MED diagnostics areas

Field	Data type
Header fields	
MORE	INTEGER
NUMBER	INTEGER
RETURNCODE	SMALLINT
Implementation-defined header field	Implementation-defined
Fields in status records	
CLASS_ORIGIN	CHARACTER VARYING (*L1*)
MESSAGE_LENGTH	INTEGER
MESSAGE_OCTET_LENGTH	INTEGER
MESSAGE_TEXT	CHARACTER VARYING (*L1*)
NATIVE_CODE	INTEGER
SQLSTATE	CHARACTER (5)
SUBCLASS_ORIGIN	CHARACTER VARYING (*L1*)
Implementation-defined status field	Implementation-defined
Where *L1* is an implementation-defined integer not less than 254.	

All diagnostics area fields in other parts of ISO/IEC 9075 that are not included in this table are not applicable to SQL/MED.

4.14.5 Null terminated strings

An input character string provided as the value of an argument to a routine may be terminated by the implementation-defined null character that terminates C character strings. If this technique is used, the the associated length argument may be set to either the length of the string excluding the null terminator or to −3, indicating NULL TERMINATION.

If NULL TERMINATION is true for the SQL-server, then all output character strings returned by the SQL-server routines are terminated by the implementation-defined null character that terminates C character strings. If NULL TERMINATION is false, then output character strings are not null terminated.

4.14.6 Null pointers

If the standard programming language of the caller of a routine supports pointers, then the caller may provide a zero-valued pointer, referred to as a *null pointer*, in the following circumstances:

— In lieu of an output argument that is to receive the length of a returned character string. This indicates that the caller wishes to prohibit the return of this information.

— In lieu of other output arguments where specifically allowed by this part of ISO/IEC 9075. This indicates that the caller wishes to prohibit the return of this information.

— In lieu of input arguments where specifically allowed by this part of ISO/IEC 9075. The semantics of such a specification depend on the context.

If the caller provides a null pointer in any other circumstances, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid use of null pointer*.

If NULL TERMINATION is false for the SQL-server, then specifying a zero buffer size for an output argument is equivalent to specifying a null pointer for that output argument.

4.14.7 SQL/MED descriptor areas

An SQL/MED descriptor area provides an interface for a description of values required for the execution of a SQL-statement by a foreign-data wrapper and for a description of values resulting from such an execution.

Each descriptor area comprises header fields and zero or more item descriptor areas. The header and item descriptor area fields are specified in Table 5, "Fields in SQL/MED descriptor areas". The header fields include a COUNT field that indicates the number of item descriptor areas.

The GetDescriptor( ) routine enables information to be retrieved from any SQL/MED descriptor area. The SetDescriptor( ) routine enables information to be set in any SQL/MED descriptor area except a WRD.

The following descriptor areas are implicitly allocated and deallocated:

— Table Reference Descriptor (TRD): This descriptor is allocated automatically by an SQL-server to describe a foreign table referenced in an SQL-statement, and is associated with a TableReferenceHandle created by an SQL-server. The foreign-data wrapper can obtain the handle of a TRD by invoking the GetTRDHandle( ) routine. It can then retrieve the information in the associated TRD descriptor by invoking the GetDescriptor( ) routine.

— Wrapper Row Descriptor (WRD): This descriptor is allocated automatically by a foreign-data wrapper to describe the result of an SQL-statement to be executed by that foreign-data wrapper, and is associated with an ExecutionHandle. The SQL-server can obtain the handle to a WRD by invoking the GetWRDHandle( ) routine. It can then retrieve the information in that WRD by invoking the GetDescriptor( ) routine.

— Server Row Descriptor (SRD): This descriptor is allocated automatically by a foreign-data wrapper and is used by the SQL-server to specify the type and location of data to be provided by the foreign-data wrapper. SRD is also associated with an ExecutionHandle. The SQL-server can obtain the handle to a SRD by invoking the GetSRDHandle( ) routine. It can then set the information in that SRD by invoking the SetDescriptor( ) routine.

— Wrapper Parameter Descriptor (WPD): This descriptor is allocated automatically by a foreign-data wrapper to describe the input values required for the execution of an SQL-statement by that foreign-data wrapper, and is associated with an ExecutionHandle. The SQL-server can obtain the handle to a WPD by invoking the GetWPDHandle( ) routine. It can then retrieve or set the information in that WPD by invoking the GetDescriptor( ) routine or SetDescriptor( ) routine, respectively.

— Server Parameter Descriptor (SPD): This descriptor is allocated automatically by a foreign-data wrapper and is used by the SQL-server to specify the type and location of input values to be provided by the SQL-server. The SPD is also associated with an ExecutionHandle. The SQL-server can obtain the handle to an SPD by invoking the GetSPDHandle( ) routine. It can then set the information in that SPD by invoking the SetDescriptor( ) routine.

Table 5—Fields in SQL/MED descriptor areas

Field	Data Type
Header fields	
COUNT	SMALLINT
DYNAMIC_FUNCTION	CHARACTER VARYING(*L*)
DYNAMIC_FUNCTION_CODE	INTEGER
KEY_TYPE	SMALLINT
TOP_LEVEL_COUNT	SMALLINT
Implementation-defined header field	Implementation-defined

Table 5—Fields in SQL/MED descriptor areas (Cont.)

Field	Data Type
	Fields in item descriptor areas
CARDINALITY	INTEGER
CHARACTER_SET_CATALOG	CHARACTER VARYING(*L*)
CHARACTER_SET_NAME	CHARACTER VARYING(*L*)
CHARACTER_SET_SCHEMA	CHARACTER VARYING(*L*)
COLLATION_CATALOG	CHARACTER VARYING(*L*)
COLLATION_NAME	CHARACTER VARYING(*L*)
COLLATION_SCHEMA	CHARACTER VARYING(*L*)
CURRENT_TRANSFORM_GROUP	CHARACTER VARYING(*L1*)
DATA	ANY
DATA_POINTER	host variable address
DATETIME_INTERVAL_CODE	SMALLINT
DATETIME_INTERVAL_PRECISION	SMALLINT
DEGREE	INTEGER
INDICATOR	INTEGER
INDICATOR_POINTER	host variable address of INTEGER
KEY_MEMBER	SMALLINT

Table 5—Fields in SQL/MED descriptor areas (Cont.)

Field	Data Type
Fields in item descriptor areas	
LENGTH	INTEGER
LEVEL	INTEGER
NAME	CHARACTER VARYING(*L*)
NULLABLE	SMALLINT
OCTET_LENGTH	INTEGER
OCTET_LENGTH_POINTER	host variable address of INTEGER
PARAMETER_MODE	SMALLINT
PARAMETER_ORDINAL_POSITION	SMALLINT
PARAMETER_SPECIFIC_CATALOG	CHARACTER VARYING(*L*)
PARAMETER_SPECIFIC_NAME	CHARACTER VARYING(*L*)
PARAMETER_SPECIFIC_SCHEMA	CHARACTER VARYING(*L*)
PRECISION	SMALLINT
RETURNED_CARDINALITY	INTEGER
RETURNED_CARDINALITY_POINTER	host variable address of INTEGER
RETURNED_OCTET_LENGTH	INTEGER
SCALE	SMALLINT
SCOPE_CATALOG	CHARACTER VARYING(*L*)
SCOPE_NAME	CHARACTER VARYING(*L*)
SCOPE_SCHEMA	CHARACTER VARYING(*L*)
SPECIFIC_TYPE_CATALOG	CHARACTER VARYING(*L*)
SPECIFIC_TYPE_NAME	CHARACTER VARYING(*L*)
SPECIFIC_TYPE_SCHEMA	CHARACTER VARYING(*L*)
TYPE	SMALLINT
UNNAMED	SMALLINT
USER_DEFINED_TYPE_CATALOG	CHARACTER VARYING(*L*)
USER_DEFINED_TYPE_NAME	CHARACTER VARYING(*L*)
USER_DEFINED_TYPE_SCHEMA	CHARACTER VARYING(*L*)
Implementation-defined SQL/MED descriptor item field	Implementation-defined

4.15 Introduction to SQL/CLI

Insert this paragraph The BuildDataLink routine can be used to build a datalink value. The GetDataLinkAttr routine can be used to extract the attributes of a datalink value.

5 Lexical elements

5.1 <SQL terminal character>

Function

Define the terminal symbols of the SQL language and the elements of strings.

Format

```
<SQL special character> ::=   !! All alternatives from ISO/IEC 9075-2
       | !! All alternatives from ISO/IEC 9075-5
       | <commercial at>
       | <dollar sign>
       | <exclamation point>

<commercial at> ::= @

<dollar sign> ::= $

<exclamation point> ::= !
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.2 <token> and <separator>

Function

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

Format

```
<non-reserved word> ::=
        !! All alternatives from ISO/IEC 9075-2
      | !! All alternatives from ISO/IEC 9075-5

      | AUTHID

      | BLOCKED

      | CONTROL

      | DB | DLCOMMENT | DLLINKTYPE
      | DLURLCOMPLETE | DLURLPATH | DLURLPATHONLY | DLURSCHEME
      | DLURLSERVER | DLVALUE

      | FILE | FS

      | INTEGRITY

      | LIBRARY | LINK

      | MAPPING

      | PASSWORD | PERMISSION

      | RECOVERY | RESTORE

      | SELECTIVE | SERVER

      | UNLINK

      | VERSION

      | WRAPPER

      | YES

      <reserved word> ::=
        !! All alternatives from ISO/IEC 9075-2
      | !! All alternatives from ISO/IEC 9075-5

      | DATALINK
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.3 Names and identifiers

Function

Specify names.

Format

```
<foreign server name> ::=
      [ <catalog name> <period> ] <unqualified foreign server name>

<foreign-data wrapper name> ::=
      [ <catalog name> <period> ] <unqualified foreign-data wrapper name>

<unqualified foreign server name> ::= <qualified identifier>

<unqualified foreign-data wrapper name> ::= <qualified identifier>
```

- 1 production deleted.

Syntax Rules

1) Insert this SR If a <foreign server name> does not contain a <catalog name>, then the <catalog name> that is specified or implicit for the SQL-client module is implicit.

2) Insert this SR If a <foreign-data wrapper name> does not contain a <catalog name>, then the <catalog name> that is specified or implicit for the SQL-client module is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR A <foreign server name> identifies a foreign server.

2) Insert this GR A <foreign-data wrapper name> identifies a foreign-data wrapper.

Conformance Rules

No additional Conformance Rules.

6 Scalar expressions

6.1 <data type>

Function

Specify a data type.

Format

```
<predefined type> ::=
         !! All alternatives from ISO/IEC 9075-2
       | !! All alternatives from ISO/IEC 9075-5
```

- 1 alternative deleted.

```
       | <datalink type>

<datalink type> ::=
       DATALINK
```

- 1 production deleted.

Syntax Rules

1) Insert this SR DATALINK specifies the data type datalink.

2) Insert this SR If <datalink length> is omitted, then an implementation-defined <datalink length> is implicit.

3) Insert this SR The maximum value of <datalink length> is implementation-defined. <datalink length> shall not be greater than this maximum value.

- 1 Rule deleted.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M001, "Datalinks", conforming SQL language shall not specify <datalink type>.

6.2 <set function specification>

Function

Specify a value derived by the application of a function to an argument.

Format

No additional Format items.

Syntax Rules

1) Insert after SR10) If the <set function specification> specifies a <set function type> that is MAX or MIN, then *DT* shall not be DATALINK or a distinct type whose source type is DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6.3 <string value function>

Function

Specify a function yielding a value of type character string or bit string.

Format

```
<string value function> ::=
        !! All alternatives from ISO/IEC 9075-2
      | !! All alternatives from ISO/IEC 9075-5
      | <datalink comment expression>
      | <link type expression>
      | <url complete expression>
      | <url path expression>
      | <url path only expression>
      | <url scheme expression>
      | <url server expression>

<datalink comment expression> ::=
      DLCOMMENT <left paren> <datalink value expression> <right paren>

<link type expression> ::=
      DLLINKTYPE <left paren> <datalink value expression> <right paren>

<url complete expression> ::=
      DLURLCOMPLETE <left paren> <datalink value expression> <right paren>

<url path expression> ::=
      DLURLPATH <left paren> <datalink value expression> <right paren>

<url path only expression> ::=
      DLURLPATHONLY <left paren> <datalink value expression> <right paren>

<url scheme expression> ::=
      DLURLSCHEME <left paren> <datalink value expression> <right paren>

<url server expression> ::=
      DLURLSERVER <left paren> <datalink value expression> <right paren>
```

Syntax Rules

1) Insert this SR The declared type of <datalink comment expression> is variable-length character string with an implementation-defined maximum length.

2) Insert this SR The declared type of <link type expression> is variable-length character string with an implementation-defined maximum length.

3) Insert this SR The declared type of <url complete expression> is variable-length character string with an implementation-defined maximum length.

4) Insert this SR The declared type of <url path expression> is variable-length character string with an implementation-defined maximum length.

5) Insert this SR The declared type of <url path only expression> is variable-length character string with an implementation-defined maximum length.

6) Insert this SR The declared type of <url scheme expression> is variable-length character string with an implementation-defined maximum length.

7) Insert this SR The declared type of <url server expression> is variable-length character string with an implementation-defined maximum length.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR Let *DVE* be the <datalink value expression> simply contained in <string value function>. Let *DV* be the result of *DVE*. If *DV* is the null value, then the result of the <string value function> is the null value.

2) Insert this GR If <datalink comment expression> is specified, then the result is the Comment of *DV*.

3) Insert this GR If <link type expression> is specified, then the result is the Link Type of *DV*.

4) Insert this GR If <url complete expression> is specified, then

Case:

a) If the Link Type of *DV* is not 'URL', then an exception condition is raised: *data exception — invalid Link Type*.

b) Otherwise,

Case:

i) If *DVE* specifies a <column reference> *CR*, then let *C* be the column identified by *CR*.

Case:

1) If the descriptor of *C* includes the link control option FILE LINK CONTROL and the read permission option READ PERMISSION DB, then the result is the File Reference of *DV* combined with an access token in an implementation-dependent manner.

2) Otherwise, the result is the File Reference of *DV*.

ii) Otherwise, the result is the File Reference of *DV*.

5) Insert this GR If <url path expression> is specified, then

Case:

a) If the Link Type of *DV* is not 'URL', then an exception condition is raised: *data exception — invalid Link Type*.

b) Otherwise,

Case:

i) If the File Reference of *DV* contains an <http url> that contains an <hpath> *HP*, then

Case:

1) If *DVE* specifies a <column reference> *CR*, then let *C* be the column identified by *CR*.

   Case:

   A) If the descriptor of *C* includes the link control option FILE LINK CONTROL and the read permission option READ PERMISSION DB, then the result is *HP* combined with an access token in an implementation-dependent manner.

   B) Otherwise, the result is *HP*.

2) Otherwise, the result is *HP*.

ii) If the File Reference of *DV* contains a <file url>, that contains the <fpath> *FP*, then

Case:

i) If *DVE* specifies a <column reference> *CR*, then let *C* be the column identified by *CR*.

   Case:

   1) If the descriptor of *C* includes the link control option FILE LINK CONTROL and the read permission option READ PERMISSION DB, then the result is *FP* combined with an access token in an implementation-dependent manner.

   2) Otherwise, the result is *FP*.

ii) Otherwise, the result is *FP*.

iii) Otherwise, a zero-length character string.

6) Insert this GR If <url path only expression> is specified, then

Case:

a) If the Link Type of *DV* is not 'URL', then an exception condition is raised: *data exception — invalid Link Type*.

b) Otherwise, the result is

   Case:

   i) If the File Reference of *DV* contains an <http url> that contains an <hpath> *HP*, then *HP*, excluding any access token.

   ii) If the File Reference of *DV* contains a <file url>, then the <fpath> contained in that <file url>.

   iii) Otherwise, a zero-length character string.

7) Insert this GR If <url scheme expression> is specified, then

Case:

a) If the Link Type of *DV* is not 'URL', then an exception condition is raised: *data exception — invalid Link Type*.

b) Otherwise, the result is

Case:

i) If the File Reference of *DV* contains an <http url>, then the <http> contained in that <http url>.

ii) If the File Reference of *DV* contains a <file url>, then the <file> contained in that <file url>.

iii) Otherwise, a zero-length character string.

8) Insert this GR If <url server expression> is specified, then

Case:

a) If the Link Type of *DV* is not 'URL', then an exception condition is raised: *data exception — invalid Link Type.*

b) Otherwise, the result is

Case:

i) If the File Reference of *DV* contains an <http url>, then the <host> contained in that <http url>.

ii) If the File Reference of *DV* contains a <file url>, then the <host> contained in that <file url>.

iii) Otherwise, a zero-length character string.

Conformance Rules

No additional Conformance Rules.

6.4 <datalink value function>

Function

Specify a function yielding a value of type datalink.

Format

```
<datalink value function> ::=
       <datalink value constructor>

<datalink value constructor> ::=
       DLVALUE <left paren> <data location> <comma> <linktype indicator>>
         [ <comma> <datalink comment> ] <right paren>

<data location> ::= <character value expression>

<linktype indicator> ::= <character value expression>

<datalink comment> ::= <character value expression>
```

Syntax Rules

1) The declared type of a <datalink value constructor> *DVC* is DATALINK.

2) If <datalink value constructor> is specified and <datalink comment> is not specified, then the <datalink comment>

   ''

   is implicit.

3) <data location>, <linktype indicator>, and, if specified, <datalink comment> shall be <character value expression>s having the same character repertoire.

Access Rules

No additional Access Rules.

General Rules

1) Let *DLOC* be the result of evaluating <data location>, let *LTYPE* be the result of evaluating <linktype indicator>, and let *CMT* be the result of evaluating <datalink comment>.

   a) If *LTYPE* or *CMT* is the null value, then an exception condition is raised: *data exception — null argument passed to datalink constructor*.

   b) If *LTYPE* is not equal to 'URL', then an exception condition is raised: *data exception — invalid data specified for datalink*.

   c) If *DLOC* is the null value, then the result of *DVC* is the null value.

d) If *DLOC* does not conform to the Format of Subclause 9.1, "URL format", then an exception condition is raised: *data exception — invalid data specified for datalink*.

• 1 subrule deleted.

e) If the number of octets occupied by the implementation-defined representation of the results of *DVC* exceeds the maximum datalink length, then an exception condition is raised: *data exception — datalink value exceeds maximum length*.

NOTE 4 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

f) Otherwise, the result of *DVC* is the datalink value *DL* such that:

i) The Link Type of *DL* is *LTYPE*.

ii) The File Reference of *DL* is *DLOC*.

iii) The Comment of *DL* is *CMT*.

2) The result of a <datalink value function> *DVF* is the result of the <datalink value constructor> contained in *DVF*.

Conformance Rules

1) Without Feature M001, "Datalinks", conforming SQL language shall not specify <datalink value function>.

6.5 <cast specification>

Function

Specify a data conversion.

Format

No additional Format items.

Syntax Rules

1) Augments SR6) If the <cast operand> is a <value expression>, then the valid combinations of *TD* and *SD* in a <cast specification> are given by the following table.

<data type> *SD* of <value expression>	<data type> of *TD*																
	EN	AN	VC	FC	VB	FB	D	T	TS	YM	DT	BO	UDT	CL	BL	RT	DL
EN	Y	Y	Y	Y	N	N	N	N	N	M	M	N	M	Y	N	N	N
AN	Y	Y	Y	Y	N	N	N	N	N	N	N	N	M	Y	N	N	N
C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	M	Y	N	N	N
B	N	N	Y	Y	Y	Y	N	N	N	N	N	N	M	Y	N	N	N
D	N	N	Y	Y	N	N	Y	N	Y	N	N	N	M	Y	N	N	N
T	N	N	Y	Y	N	N	N	Y	Y	N	N	N	M	Y	N	N	N
TS	N	N	Y	Y	N	N	Y	Y	Y	N	N	N	M	Y	N	N	N
YM	M	N	Y	Y	N	N	N	N	N	Y	N	N	M	Y	N	N	N
DT	M	N	Y	Y	N	N	N	N	N	N	Y	N	M	Y	N	N	N
BO	N	N	Y	Y	N	N	N	N	N	N	N	Y	M	Y	N	N	N
UDT	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	N	N
BL	N	N	N	N	N	N	N	N	N	N	N	N	M	N	Y	N	N
RT	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N
DL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y

Where:

EN = Exact Numeric
AN = Approximate Numeric
C = Character (Fixed- or Variable-length, or character large object)
FC = Fixed-length Character
VC = Variable-length Character
CL = Character Large Object
B = Bit String (Fixed- or Variable-length)
FB = Fixed-length Bit String
VB = Variable-length Bit String
D = Date
T = Time
TS = Timestamp
YM = Year-Month Interval
DT = Day-Time Interval
BO = Boolean
UDT = User-Defined Type
BL = Binary Large Object
RT = Reference type
DL = Datalink

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR If *TD* and *SD* are datalink types, then *TV* is *SV*.

Conformance Rules

No additional Conformance Rules.

6.6 <value expression>

Function

Specify a value.

Format

```
<value expression> ::=
        !! All alternatives from ISO/IEC 9075-2
      | !! All alternatives from ISO/IEC 9075-5
      | <datalink value expression>
```

Syntax Rules

1) Replaces SR1) The declared type of a <value expression> is the declared type of the <numeric value expression>, <string value expression>, <datetime value expression>, <interval value expression>, <boolean value expression>, <user-defined type value expression>, <row value expression>, <collection value expression>, <reference value expression>, or <datalink value expression>, respectively.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6.7 <datalink value expression>

Function

Specify a datalink value.

Format

```
<datalink value expression> ::=
        <datalink value function>
      | <value expression primary>
```

Syntax Rules

1) The declared type of <value expression primary> shall be DATALINK.

Access Rules

None.

General Rules

1) Case:

   a) If <datalink value function> *DVF* is specified, then the result of the <datalink value expression> is the result of *DVF*.

   b) If <value expression primary> *VEP* is specified, then the result of the <datalink value expression> is the result of *VEP*.

Conformance Rules

1) Without Feature M001, "Datalinks", conforming SQL language shall not specify <datalink value expression>.

7 Query expressions

7.1 <table reference>

Function

Reference a table.

Format

No additional Format items

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR2) If the <table reference> simply contains a <table name> *TN*, then

Case:

a) If *TN* identifies a view or a base table *T*, then

Case:

i) If ONLY is specified, then the <table reference> references the table that consists of every row in *T*, except those rows that have a subrow in a proper subtable of *T*.

ii) Otherwise, the <table reference> references the table that consists of every row of *T*.

b) If *TN* identifies a foreign table *FTN*, then the table referenced by the <table reference> is effectively determined as follows:

i) Let *FSN* be the name of the foreign server included in the table descriptor of the foreign table identified by *FTN*. Let *WN* be the name of the foreign-data wrapper included in the foreign server descriptor of the foreign server identified by *FSN*. Let *WR* be the foreign-data wrapper identified by *WN*. Let *WRLN* be the name of the library identified in the foreign-data wrapper descriptor of *WR*.

ii) Case:

1) If the current SQL-session context includes a {foreign-data wrapper name : WrapperEnvHandle} pair whose foreign-data wrapper name is equivalent to *WN*, then let *WEH* be the WrapperEnvHandle associated with *WN*.

2) Otherwise:

A) Let *WH* be the WrapperHandle allocated for the foreign-data wrapper identified by *WN*. The resource identified by *WH* is referred to as an *allocated foreign-data wrapper description*.

B) Let *WEH* be the WrapperEnvHandle returned by the invocation of AllocWrapperEnv routine in the library identified by *WRLN*, with *WH* as the argument).

C) The {*WN* : *WEH*} pair is included in the current SQL session context.

D) *WH* is deallocated and all its resources are freed.

iii) Case:

1) If the current SQL-session context includes a {foreign server name : FSConnectionHandle} pair whose foreign server name is equivalent to *FSN*, then let *FSCH* be the FSConnectionHandle associated with *FSN*.

2) Otherwise:

A) Let *SH* be the ServerHandle allocated for the foreign server identified by *FSN*. The resource identified by *SH* is referred to as an *allocated foreign server description*.

B) Let UH be the UserHandle allocated for the user mapping identified by the current authorization identifier. The resource identified by *UH* is referred to as an *allocated user mapping description*.

C) Let *FSCH* be the FSConnectionHandle returned by the invocation of the ConnectForeignServer routine in the library identified by *WRLN* with *WEH*, *SH*, and *UH* as the arguments.

D) The {*FSN* : *FSCH*} pair is included in the current SQL session context.

E) *SH* is deallocated and all its resources are freed.

F) *UH* is deallocated and all its resources are freed.

iv) Let *RQH* be the RequestHandle allocated for a <query specification> of the form "`SELECT * FROM FTN`".

v) Let *TRH* be the TableReferenceHandle allocated for a <table reference> of the form "*FTN*".

vi) Let *N* be the number of columns of the foreign table *FT* identified by *FTN*. Let CN_i, 1 (one) $\leq i \leq N$, be the column name of *i*-th column of *FT*. Let VEH_i, 1 (one) $\leq i \leq N$ be the *i*-th ValueExpressionHandle allocated for a <column reference> of the form "CN_i.

vii) A table reference descriptor *TRD* is automatically allocated. Each of the fields in *TRD* that have non-blank entries in Table 37, "SQL/MED descriptor field default values", are set to the specified default values. All other fields in *TRD* are initially undefined.

viii) The General Rules of Subclause 21.4, "Implicit DESCRIBE OUTPUT USING clause", are applied with '`SELECT * FROM` *FTN*' and *TRD* as *SOURCE* and *DESCRIPTOR*, respectively.

ix) Let *TRDH* be the TableReferenceDescriptorHandle allocated for *TRD*. *TRDH* is associated with *TRH*.

x) Let *RPH* and *EXH* be the ReplyHandle and ExecutionHandle, respectively, returned by the invocation of InitForeignRequest routine in the library identified by *WRLN* with *WEH*, *FSCH*, and *RQH* as arguments.

xi) Let *NRTR* be the NumberOfTableReferences that would be returned by an invocation of the GetNumberOfReplyTableReferences( ) routine with *RPH* as the ReplyHandle parameter.

xii) Let TRN_i", 1 (one) $\leq i \leq NRTR$, be the TableReferenceNumber that would be returned by an invocation of the GetReplyTableReference( ) routine with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.

xiii) Let *NSLE* be the NumberOfSelectListElements that would be returned by an invocation of the GetNumberOfReplySelectListElements( ) routine with *RPH* as the ReplyHandle parameter.

xiv) Let $SELN_i$, 1 (one) $\leq i \leq NSLE$, be the SelectListElementNumber that would returned by an invocation of the GetReplySelectListElement( ) routine with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.

xv) VEH_i, 1 (one) $\leq i \leq N$, is deallocated and all its resources are freed.

xvi) *TRH* is deallocated and all its resources are freed.

xvii) *RQH* is deallocated and all its resources are freed.

xviii) The FreeReplyHandle( ) routine in the library identified by *WRLN* is invoked with RPH as the argument.

xix) Let *WRD* be the WRDHandle that would be returned by an invocation of the GetWRDHandle( ) routine with *EXH* as the ExecutionHandle parameter.

xx) Let *NC* be the value of the COUNT descriptor field that would be returned by invocation of the GetDescriptor( ) routine with *WRD* as the DescriptorHandle parameter, 0 (zero) as the RecordNumber parameter, and the code for COUNT from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

xxi) Let DT_j be the effective data type of the *j*-th column, for 1 (one) $\leq j \leq NC$, as represented by the values of the TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields that would be returned by separate invocations of the GetDescriptor( ) routine with *WRD* as the DescriptorHandle parameter, *j* as the RecordNumber parameter, and the code for the fields TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

xxii) Let *SRD* be the SRDHandle that would be returned by an invocation of the GetSRDHandle routine with *EXH* as the ExecutionHandle parameter.

xxiii) Let TDT_j be the effective data type of the *j*-th <target specification>, for 1 (one) ≤ $j \leq NC$, as represented by the values of the TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields that would be set by separate invocations of the SetDescriptor( ) routine with *SRD* as the DescriptorHandle parameter, *j* as the RecordNumber parameter, and the code for the fields TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

xxiv) Let *WPD* be the WPDHandle that would be returned by an invocation of the GetWPDHandle( ) routine with *EXH* as the ExecutionHandle parameter.

xxv) Let *SPD* be the SPDHandle that would be returned by an invocation of the GetSPDHandle( ) routine with *EXH* as the ExecutionHandle parameter.

xxvi) Let *NP* be the value of the COUNT descriptor field that would be returned by invocation of the GetDescriptor( ) routine with WPD as the DescriptorHandle parameter, 0 (zero) as the RecordNumber parameter, and the code for COUNT from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

xxvii) If *NP* is greater than zero, then:

i) Let PDT_j be the effective data type of the *j*-th <dynamic parameter specification>, for 1 (one) ≤ $j \leq NP$, as represented by the values of the TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields that would be retrieved by separate invocations of the GetDescriptor( ) routine with *WPD* as the DescriptorHandle parameter, *j* as the RecordNumber parameter, and the code for the fields TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

ii) Let SDT_j be the effective data type of the *j*-th <value specification>, for 1 (one) ≤ $j \leq NP$, as represented by the values of the TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields that would be set by separate invocations of

the SetDescriptor( ) routine with *SPD* as the DescriptorHandle parameter, *j* as the RecordNumber parameter, and the code for the fields TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

xxviii) The Open( ) routine in the library identified by *WRLN* is invoked with *EXH* as the argument.

xxix) The <table reference> references the table that consists of every row returned by the repeated invocation of Iterate routine in the library identified by *WRLN* with *EXCH* as the argument until the return code indicates **No data found**.

xxx) The Close( ) routine in the library identified by *WRLN* is invoked with *EXH* as the argument.

xxxi) The FreeExecutionHandle( ) routine in the library identified by *WRLN* is invoked with *EXH* as the argument.

Conformance Rules

No additional Conformance Rules.

7.2 <joined table>

Function

Specify a table derived from a Cartesian product, inner or outer join, or union join.

Format

No additional Format items.

Syntax Rules

1) Insert after SR1) No <column name> contained in a <join column list> shall identify a column whose declared type is DATALINK or a distinct type whose source type is DATALINK.

2) Insert after SR7)c) The declared type of neither C_1 nor C_2 shall be DATALINK or a distinct type whose source type is DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

7.3 <group by clause>

Function

Specify a grouped table derived by the application of the <group by clause> to the result of the previously specified clause.

Format

No additional Format items.

Syntax Rules

1) Insert after SR5) The declared type of a grouping column shall not be DATALINK, be based on DATALINK, or be a distinct type whose source type is DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

7.4 <query expression>

Function

Specify a table.

Format

No additional Format items.

Syntax Rules

1) Replace GR21) If the declared type of any column of a <query term> is any of the following, then ALL shall be specified:

   a) A large object string type.

   b) DATALINK, based on DATALINK, or a distinct type whose source type is DATALINK.

2) Replace SR22) If the declared type of any column of a <query primary> is any of the following, then ALL shall be specified:

   a) A large object string type.

   b) DATALINK, based on DATALINK, or a distinct type whose source type is DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

8 Predicates

8.1 <comparison predicate>

Function

Specify a comparison of two row values.

Format

No additional Format items.

Syntax Rules

1) Insert after SR5)a) If the declared type of X_i or Y_i is a datalink, then <comp op> shall be either <equals operator> or <not equals operator>.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR1)b)iii) If the declared types of *XV* and *YV* are datalink types, then *X* is equal to *Y* if and only if DLCOMMENT(*X*) is equal to DLCOMMENT(*Y*) and DLLINKTYPE(*X*) is equal to DLLINKTYPE(*Y*) and DLURLPATHONLY(*X*) is equal to DLURLPATHONLY(*Y*) and DLURLSCHEME(*X*) is equal to DLURLSCHEME(*Y*) and DLURLSERVER(*X*) is equal to DLURLSERVER(*Y*).

Conformance Rules

No additional Conformance Rules.

9 URLs

9.1 URL format

Function

Specify the precise format of a URL within a datalink. The specification is a direct translation of the format of HTTP and FILE URLs specified in RFC 1738, as modified by RDC 1808, except that "localhost" has been omitted from the format of FILE URL. RFC 1738, RFC 1808, and RFC 2368 specify other URL schemes; URLs formatted according to those other schemes are not supported within datalinks.

Format

```
<url> ::=
          <http url>
        | <file url>

<http url> ::= <http> <colon> <solidus> <solidus> <host port> [ <solidus> <hpath> ]

<http> ::= { h | H }{ t | T } { t | T } { p | P }

<host port> ::= <host> [ <colon> <port> ]

<host> ::=
          <host name>
        | <host number>

<host name> ::= [ { <domain label> <period> }... ] <top label>

<domain label> ::=
          <letter or digit>
        | <letter or digit> <label tail>

<letter or digit> ::=
          <simple Latin letter>
        | <digit>

<label tail> ::= [ { <letter or digit> | <minus sign> }... ] <letter or digit>

<top label> ::=
          <simple Latin letter>
        | <simple Latin letter> <label tail>

<host number> ::= <digits> <period> <digits> <period> <digits> <period> <digits>

<digits> ::= <digit>...

<port> ::= <digits>

<hpath> ::= <hsegment> [ { <solidus> <hsegment> }... ]

<hsegment> ::= [ <hsegment character>... ]
```

9.1 URL format

```
<hsegment character> ::=
        <uchar>
```

- 1 alternative deleted.

```
       | <colon>
       | <commercial at>
       | <ampersand>
       | <equals operator>

<uchar> ::=
        <unreserved>
       | <escape>

<unreserved> ::=
        <simple Latin letter>
       | <digit>
       | <safe>
       | <extra>

<safe> ::=
        <dollar sign>
       | <minus sign>
       | <underscore>
       | <period>
       | <plus sign>

<extra> ::=
        <exclamation point>
       | <asterisk>
       | <quote>
       | <left paren>
       | <right paren>
       | <comma>

<escape> ::= <percent> <hexit> <hexit>

<file url> ::= <file> <colon> <solidus> <solidus> <host> <solidus> <fpath>

<file> ::= { f | F } { i | I } { l | L } { e | E }

<fpath> ::= <fsegment> [ { <solidus> <fsegment> }... ]

<fsegment> ::= [ <fsegment character>... ]

<fsegment character> ::=
        <uchar>
       | <question mark>
       | <colon>
       | <commercial at>
       | <ampersand>
       | <equals operator>
```

Syntax Rules

None.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

10 Data assignment rules and routine determination

10.1 Retrieval assignment

Function

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

Syntax Rules

1) Insert this SR If the declared type of *T* is DATALINK, then the declared type of *V* shall be DATALINK.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR4)u) If the declared type of *T* is DATALINK, then the value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

10.2 Store assignment

Function

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

Syntax Rules

1) Insert this SR If the declared type of *T* is DATALINK, then the declared type of *V* shall be DATALINK.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR2)b)xxi) If the declared type of *T* is DATALINK, then the value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

10.3 Data types of results of aggregations

Function

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

Syntax Rules

1) Insert after SR3)h) If any data type in *DTS* is DATALINK, then each data type in *DTS* shall be DATALINK and the result data type is DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.4 Type precedence list determination

Function

Determine the type precedence list of a given type.

Syntax Rules

1) Insert this SR If *DT* is datalink, then *TPL* is

```
DATALINK DT
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11 Additional common elements

11.1 <privileges>

Function

Specify privileges.

Format

```
<privileges> ::=
        !! All alternatives from ISO/IEC 9075-2
      | <foreign-data wrapper name>
      | <foreign server name>
```

Syntax Rules

1) Augment SR3) Add <foreign server name> and <foreign-data wrapper name> to the list of <object name>s that shall require the specification of USAGE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.2 <generic options>

Function

Specify a list of options identified by keywords.

Format

```
<generic options> ::=
       OPTIONS <left paren> <generic option list> <right paren>

<generic option list> ::=
       <generic option> [ { <comma> <generic option> }... ]

<generic option> ::= <option name> [ <option value> ]

<option name> ::= <regular identifier>

<option value> ::= <character string literal>
```

Syntax Rules

1) The permissible <option name>s are implementation-defined.

2) The permissible <option value>s are implementation-defined.

Access Rules

None.

General Rules

1) A generic options descriptor *GOPD* is created as follows. Let *n* be the number of <generic option>s contained in <generic option list> *GOPL*. For *i* ranging from 1 (one) to *n*, the *i*-th <option name> included in *GOPD* is the *i*-th <option name> contained in *GOPL* and the *i*-th option value included in *GOPD* is the *i*-th <option value> contained in *GOPL*, if any.

Conformance Rules

None.

11.3 <alter generic options>

Function

Change the contents of a generic options descriptor

Format

```
<alter generic options> ::=
       OPTIONS <left paren> <alter generic option list> <right paren>

<alter generic option list> ::=
       <alter generic option> [ { <comma> <alter generic option> }... ]

<alter generic option> ::=
       [ <alter operation> ] <option name> [ <option value> ]

<alter operation> ::=
         ADD
       | SET
       | DROP
```

Syntax Rules

1) Let *GOPD* be the applicable generic options descriptor. Let *AGOPL* be the <alter generic option list>.

2) For each <alter generic option> *AGOP* contained in *AGOPL*, if <alter operation> is omitted, then *ADD* is implicit. Let *AOP* and *OPN* be the <alter operation> and <option name>, respectively, specified or implied by *AGOP*.

   Case:

   a) If *AOP* is ADD, then <option value> shall be specified and *GOPD* shall not include an <option name> that is equivalent to *OPN*.

   b) Otherwise, <option value> shall be specified and *GOPD* shall include an <option name> that is equivalent to *OPN*.

Access Rules

None.

General Rules

1) For each <alter generic option> *AGOP* contained in *AGOL*, let *AOP* and *OPN* be the <alter operation> and <option name>, respectively, specified or implied by *AGOP* and let *OPV* be the result of <option value> contained in *AGOP*.

   Case:

   a) If *AOP* is ADD, then let *n* be the number of <option name>s included in *GOPD*. *OPN* is added as the *n*+1-th <option name> included in *GOPD* and *OPV* is added as the *n*+1-th <option value> included in *GOPD*.

b) If *AOP* is SET, then let *i* be the ordinal position of *OPN* in *GOPD*. The *i*-th <option value> in *GOPD* is replaced by *OPV*.

c) If *AOP* is DROP, then let *i* be the ordinal position of *OPN* in *GOPD*. The *i*-th <option name> and the *i*-th <option value> are removed from *GOPD*.

Conformance Rules

None.

12 Schema definition and manipulation

12.1 <schema definition>

Function

Define a schema.

Format

```
<schema element> ::=
         !! All alternatives from ISO/IEC 9075-2
       | !! All alternatives from ISO/IEC 9075-5
```

- 1 alternative deleted.

```
       | <foreign table definition>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

12.2 <table definition>

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

Format

```
<column option list> ::=   !! All alternatives from ISO/IEC 9075-2
       | !! All alternatives from ISO/IEC 9075-5
       [ <datalink control definition> ]
```

Syntax Rules

1) Insert after SR9)d) If *CO* specifies <datalink control definition> *DCS*, then let *CURITIBA* be the <column name> contained in *RCD* followed in turn by the <data type> or <domain name> contained in *RCD*, the <default clause> (if any) contained in *RCD*, every <column constraint definition> contained in *RCD*, and *DCS*. *RCD* is replaced by *CURITIBA*.

2) Replace SR17) A <column option list> shall immediately contain either a <scope clause> or a <default clause>, or at least one <column constraint definition>, or a <collate clause>, or a <datalink control definition>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

12.3 <column definition>

Function

Define a column of a base table.

Format

```
<column definition> ::=
        !! All alternatives from ISO/IEC 9075-2
      [ <datalink control definition> ]

<datalink control definition> ::=
        NO LINK CONTROL
      | FILE LINK CONTROL <datalink file control option>

<datalink file control option> ::=
      <integrity option> <read permission> <write permission>
      <recovery option> [ <unlink option> ]

<integrity option> ::=
        INTEGRITY ALL
      | INTEGRITY SELECTIVE

<read permission> ::=
        READ PERMISSION FS
      | READ PERMISSION DB

<write permission> ::=
        WRITE PERMISSION FS
      | WRITE PERMISSION BLOCKED

<recovery option> ::=
        RECOVERY NO
      | RECOVERY YES

<unlink option> ::=
        ON UNLINK RESTORE
      | ON UNLINK DELETE
```

Syntax Rules

1) Insert this SR If <datalink control definition> is specified, then <data type> shall specify either DATALINK or the <user-defined type> of a distinct type whose source data type is DATALINK.

   NOTE 5 – "Source data type" is defined in Subclause 11.40, "<user-defined type definition>", in ISO/IEC 9075-2.

2) Insert this SR If <data type> specifies DATALINK or the <user-defined type> of a distinct type whose source data type is DATALINK and <datalink control definition> is not specified, then NO LINK CONTROL is implicit.

   NOTE 6 – "Source data type" is defined in Subclause 11.40, "<user-defined type definition>", in ISO/IEC 9075-2.

3) Insert this SR If FILE LINK CONTROL is specified, then:

a) If INTEGRITY SELECTIVE is specified, then READ PERMISSION FS, WRITE PERMISSION FS, and RECOVERY NO shall be specified.

b) If READ PERMISSION DB is specified, then WRITE PERMISSION BLOCKED shall be specified.

c) If WRITE PERMISSION BLOCKED is specified, then INTEGRITY ALL and <unlink option> shall be specified.

d) If WRITE PERMISSION FS is specified, then READ PERMISSION FS and RECOVERY NO shall be specified and <unlink option> shall not be specified.

e) If RECOVERY YES is specified, then WRITE PERMISSION BLOCKED shall be specified.

f) If UNLINK DELETE is specified, then READ PERMISSION DB shall be specified.

NOTE 7 – Valid combinations of <datalink file control option> resulting from this Syntax Rule are shown in Table 2, "Valid datalink file control options".

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR5)f) Inclusions specific to columns of declared type DATALINK, as follows:

a) The link control, according to whether NO LINK CONTROL or FILE LINK CONTROL is specified.

b) If FILE LINK CONTROL is specified, then:

i) The integrity control option, according to whether INTEGRITY ALL or INTEGRITY SELECTIVE is specified.

ii) The read permission option, according to whether READ PERMISSION FS or READ PERMISSION DB is specified.

iii) The write permission option, according to whether WRITE PERMISSION FS or WRITE PERMISSION BLOCKED is specified.

iv) The recovery option, according to whether RECOVERY NO or RECOVERY YES is specified.

v) The unlink option, according to whether ON UNLINK RESTORE or ON UNLINK DELETE is specified.

c) If NO LINK CONTROL is specified, then:

i) The integrity control option is NONE.

ii) The read permission option is FS.

iii) The write permission option is FS.

iv) The recovery option is NO.

v) The unlink option is NONE.

Conformance Rules

No additional Conformance Rules.

12.4 <unique constraint definition>

Function

Specify a uniqueness constraint for a table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR1) The declared type of no column identified by any <column name> in the <unique column list> shall be DATALINK, based on DATALINK, or a distinct type whose source type is DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

12.5 <drop column definition>

Function

Destroy a column of a base table.

Format

No additional Format items

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR4) If the declared type of *C* is DATALINK or some distinct type with a source data type of DATALINK, whose descriptor includes a <datalink control definition> that specifies FILE LINK CONTROL, then, for every non-null value *DLV* in *C*, let *EF* be the external file referenced by *DLV*.

   Case:

   a) If *EF* is linked, then *EF* is unlinked.

      NOTE 8 – The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in Subclause 4.9, "Columns, fields, and attributes".

   b) Otherwise, an exception condition is raised: *datalink exception — external file not linked*.

Conformance Rules

No additional Conformance Rules.

- 2 Subclauses deleted.

12.6 <foreign table definition>

Function

Define a foreign table.

Format

```
<foreign table definition> ::=
       CREATE FOREIGN TABLE <table name>
         [ <left paren> <basic column definition list> <right paren> ]
         SERVER <foreign server name> [ <table generic options> ]

<table generic options> ::= <generic options>

<basic column definition list> ::=
       <basic column definition> [ <comma> <basic column definition>... ]

<basic column definition> ::=
       <column name> <data type> [ <column generic options> ]

<column generic options> ::= <generic options>
```

Syntax Rules

1) If <foreign table definition> is contained in a <schema definition>, and if the <table name> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>.

2) Let *TN* be the <table name>. Let *S* be the schema identified by the explicit or implicit schema name of *TN*. *S* shall not include a table descriptor whose table name is equivalent to *TN*.

3) If <basic column definition list> is specified, then let *n* be the cardinality of the <basic column definition list>. For all *i*, 1 (one) $\leq i \leq n$:

   a) For all *j*, 1 (one) $\leq j \leq n$, if the <column name> contained in the *i*-th <basic column definition> is equivalent to the <column name> contained in the *j*-th <basic column definition>, then *i*=*j*.

   b) If the <data type> contained in the *i*-th <basic column definition> specifies a <character string type> and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification> of the <schema definition> that created the schema *S* is implicit.

4) Let *FSN* be the <foreign server name>.

5) The catalog identified by the explicit or implicit catalog name of *FSN* shall include a foreign server descriptor whose foreign server name is equivalent to *FSN*.

6) If the <foreign table definition> is contained in a <schema definition> *SD*, then let *A* be the explicit or implicit <authorization identifier> of *SD*. Otherwise, let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *TN*.

Access Rules

1) If <foreign table definition> is contained in an SQL-client module, then the enabled authorization identifiers shall include *A*.

2) If <basic column definition list> is specified, then for each <data type> *DT* simply contained in <basic column definition list>, if *DT* is one of the following:

   a) A user-defined type *U*.

   b) A reference type whose referenced type is a user-defined type *U*.

   c) An array type whose element type is a user-defined type *U*.

   d) An array type whose element type is a reference type whose referenced type is a user-defined type *U*.

   e) A row type with a field that has a declared type that is:

      i) A user-defined type *U*.

      ii) A reference type whose referenced type is a user-defined type *U*.

      iii) An array type whose element type is a user-defined type *U*.

      iv) An array type whose element type is a reference type whose referenced type is a user-defined type *U*.

   then the applicable privileges of *A* shall include USAGE on *U*.

****Editor's Note****
Don't we want a rule allowing implementation-defined access rules for creating a foreign table? Also, we may want to require a USAGE privilege for foreign servers, since creating a foreign table implies a dependency on a foreign server (which in turn depends on a foreign-data wrapper). See Possible Problem **MED-020**.

3) The applicable privileges shall include the USAGE privilege on the foreign-server identified by <foreign-server name>.

General Rules

1) A foreign table descriptor *FTD* is created in *S*. *FTD* includes:

   a) The table name *TN*.

   b) The foreign server name *FSN*.

   c) If <table generic options> *TGO* is specified, then the generic options descriptor created by *TGO*; otherwise, an empty generic options descriptor.

   d) Case:

      i) If <basic column definition list> *BCDL* is specified, then *n* column descriptors. For each <basic column definition> BCD_i, 1 (one) $\leq i \leq n$, the corresponding *i*-th column

descriptor includes:

1) The <column name> contained in BCD_i.

2) An indication that the column name is not an implementation-dependent name.

3) The data type descriptor of the <data type> *DT* simply contained in BCD_i.

4) The ordinal position, *i*.

5) The implementation-defined nullability characteristic.

6) The implementation-defined <default option>.

7) If <column generic options> *CGO* is specified, then the generic options descriptor created by *CGO*; otherwise, an empty generic options descriptor.

ii) Otherwise, the column descriptors included in *FTD* are implementation-defined.

e) An indication that the table is not referenceable.

f) An empty list of direct supertable names.

g) An empty list of direct subtable names.

2) Let *T* be the table described by *FTD*. Let *m* be the number of column descriptors CD_i, 1 (one) $\leq i \leq m$, included in *FTD*. The row type of *T* consists of *m* fields F_i such that, for all *i*, 1 (one) $\leq i \leq m$, the field name of F_i is the column name included in CD_i and the declared type of F_i is the data type described by the data type descriptor included in CD_i.

3) A set of privilege descriptors, each of whose grantor is set to the special grantor value "_SYSTEM", is created that define the privileges on *T* and on the columns of *T*. These privileges effectively define the same privileges that the user mapping of *A* has on the source of the foreign table on the foreign server identified by *FSN*. These privileges are grantable if the effective same privileges on the source of the foreign table are grantable.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <foreign table definition>.

12.7 <alter foreign table statement>

Function

Change the definition of a foreign table.

Format

```
<alter foreign table statement> ::=
       ALTER FOREIGN TABLE <table name> <alter foreign table action>

<alter foreign table action> ::=
         <add basic column definition>
       | <alter basic column definition>
       | <drop basic column definition>
       | <alter generic options>
```

Syntax Rules

1) The schema *S* identified by the explicit or implicit schema name of the <table name> *TN* shall include a foreign table descriptor *FTD* whose table name is equivalent to *TN*. *FTD* is the descriptor of the foreign table being altered.

2) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *TN*.

3) If <alter generic options> *AGO* is specified, then the Syntax Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *FTD* as the applicable generic options descriptor.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

1) *FTD* is modified as specified by <alter foreign table action>.

2) If <alter generic options> *AGO* is specified, then the General Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *FTD* as the applicable generic options descriptor.

3) If <alter generic options> is specified, any effect on *FTD*, apart from that on its generic options descriptor, is implementation-defined.

4) Let *T* be the table described by *FTD*. Let *m* be the number of column descriptors CD_i, 1 (one) $\leq i \leq m$, included in *FTD*. The row type of *T* consists of *m* fields F_i such that, for all *i*, 1 (one) $\leq i \leq m$, the field name of F_i is the column name included in CD_i and the declared type of F_i is the data type described by the data type descriptor included in CD_i.

Conformance Rules

1) Without Feature M002, “Foreign-data wrapper”, conforming SQL language shall not specify <alter foreign table statement>.

12.8 <add basic column definition>

Function

Add a column to a foreign table.

Format

```
<add basic column definition> ::=
       ADD [ COLUMN ] <basic column definition>
```

Syntax Rules

1) Let *FTD* be the descriptor of the foreign table being altered.

2) *FTD* shall not include a column descriptor whose column name is equivalent to the <column name> *CN* specified in the <basic column definition> *BCD*.

3) Let *A* be the <authorization identifier> that owns the schema that includes *FTD*.

Access Rules

1) Let *DT* be the <data type> simply contained in *BCD*. If *DT* is one of the following:

   a) A user-defined type *U*.

   b) A reference type whose referenced type is a user-defined type *U*.

   c) An array type whose element type is a user-defined type *U*.

   d) An array type whose element type is a reference type whose referenced type is a user-defined type *U*.

   e) A row type with a field that has a declared type that is:

      i) A user-defined type *U*.

      ii) A reference type whose referenced type is a user-defined type *U*.

      iii) An array type whose element type is a user-defined type *U*.

      iv) An array type whose element type is a reference type whose referenced type is a user-defined type *U*.

   then the applicable privileges of *A* shall include USAGE on *U*.

General Rules

1) Let *n* be the number of column descriptors included in *FTD*.

2) The degree of the table being altered by the containing <alter foreign table statement> is increased by 1 (one).

3) A column descriptor *CD* is added to *FTD*. *CD* includes:

   a) The <column name> *CN* contained in *BCD*.

b) An indication that the column name is not an implementation-dependent name.

c) The data type descriptor of the <data type> *DT* simply contained in *BCD*.

d) The ordinal position, *n*+1.

e) The implementation-defined nullability characteristic.

f) The implementation-defined <default option>.

g) If <column generic options> *CGO* is specified, then the generic options descriptor created by *CGO*; otherwise, an empty generic options descriptor.

4) For every table privilege descriptor that specifies *T* and a privilege of SELECT, a new column privilege descriptor is created that specifies *T*, the same action, grantor, and grantee, and the same grantability, and specifies *CN*.

Conformance Rules

None.

12.9 <alter basic column definition>

Function

Change the definition of a column pf a foreign table.

Format

```
<alter basic column definition> ::=
       ALTER [ COLUMN ] <column name> <alter basic column action>

<alter basic column action> ::=
       <alter generic options>
```

Syntax Rules

1) Let *FTD* be the descriptor of the foreign table identified in the containing <alter table statement>.

2) *FTD* shall include a column descriptor *CD* whose column name is equivalent to <column name>.

3) Let *C* be the column described by *CD*.

Access Rules

None.

General Rules

1) *CD* is modified as specified by <alter column action>.

2) If <alter generic options> is specified, any effect on *CD*, apart from that on its generic options descriptor, is implementation-defined.

Conformance Rules

None.

12.10 <drop basic column definition>

Function

Destroy a column of a foreign table.

Format

```
<drop basic column definition> ::=
       DROP [ COLUMN ] <column name> <drop behavior>
```

Syntax Rules

1) Let *FTD* be the descriptor of the foreign table being altered.

2) *FTD* shall include a column descriptor *CD* whose column name is equivalent to the <column name> *CN*.

3) *FTD* shall include at least two column descriptors.

4) Let *C* be the column described by *CD*.

5) 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.

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

   d) Either an explicit trigger column list or a triggered action column set of any trigger descriptor.

   NOTE 9 – A <drop column definition> that does not specify CASCADE will fail if there are any references to that column resulting from the use of CORRESPONDING, NATURAL, or SELECT * (except where contained in an exists predicate>).

   NOTE 10 – If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

   NOTE 11 – *CN* may be contained in an implicit trigger column list of a trigger descriptor.

Access Rules

None.

General Rules

1) Let *TR* be the trigger name of any trigger descriptor having an explicit trigger column list or a triggered action column set that contains *CN*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

   DROP TRIGGER *TR*

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

```
REVOKE SELECT(CN) ON TABLE TN FROM A CASCADE
```

3) Let *R* be any SQL-invoked routine whose routine descriptor contains CN in the <SQL routine body>. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed for every *R* without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

4) *CD* is destroyed and the ordinal position of every column descriptor following *CD* in *FTD* is reduced by 1 (one).

5) The degree of the table described by *FTD* is reduced by 1 (one).

Conformance Rules

None.

12.11 <drop foreign table statement>

Function

Destroy a foreign table.

Format

```
<drop foreign table statement> ::=
       DROP FOREIGN TABLE <table name> <drop behavior>
```

Syntax Rules

1) The schema *S* identified by the explicit or implicit schema name of the <table name> *TN* shall include a foreign table descriptor *FTD* whose table name is equivalent to *TN*. Let *T* be the table described by *FTD*.

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

   c) The trigger action of any trigger descriptor.

   NOTE 12 – If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns *S*.

General Rules

1) Every row of *T* is effectively deleted at the end of the SQL-statement, prior to the checking of any integrity constraints.

   NOTE 13 – This deletion creates neither a new trigger execution context nor the definition of a new state change in the current trigger execution context.

2) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *TN* FROM *A* CASCADE

3) Let *R* be any SQL-invoked routine whose routine descriptor contains *TN* in the <SQL routine body>. 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) *FTD* is destroyed.

Conformance Rules

1) Without Feature M002, “Foreign-data wrapper”, conforming SQL language shall not specify <drop foreign table statement>.

13 Foreign server and foreign-data wrapper definition and manipulation

13.1 <foreign server definition>

Function

Define a foreign server.

Format

```
<foreign server definition> ::=
       CREATE SERVER <foreign server name>
       [ TYPE <server type> ]
       [ VERSION <server version> ]
       [ AUTHORIZATION <authorization identifier> ]
       FOREIGN DATA WRAPPER <foreign-data wrapper name>
```

- 1 option deleted.

```
       [ <generic options> ]

<server type> ::= !! See the Syntax Rules

<server version> ::= !! See the Syntax Rules
```

- 1 production deleted.

Syntax Rules

1) Let *FSN* be the <foreign server name>. Let *C1* be the catalog identified by the explicit or implicit catalog name of *FSN*. *C1* shall not include a foreign server descriptor whose foreign server name is equivalent to *FSN*.

2) Let *WN* be the <foreign-data wrapper name>. Let *C2* be the catalog identified by the explicit or implicit catalog name of *WN*. *C2* shall include a foreign-data wrapper descriptor whose foreign-data wrapper name is *WN*.

3) Case:

   a) If the <foreign server definition> is contained in an SQL-client module that has a <module authorization identifier> specified, then an <authorization identifier> equal to that <module authorization identifier> is implicit for the <foreign server definition>.

   b) Otherwise, an <authorization identifier> equal to the SQL-session user identifier is implicit.

4) The permissible Format and values for <server type> and <server version> are implementation-defined.

Access Rules

1) The applicable privileges shall include the USAGE privilege on the foreign-data wrapper identified by <foreign-data wrapper name>.

****Editor's Note****

The access rules for <foreign server definition> are completely implementation-defined. Yet creating a foreign server imposes a dependency on a foreign-data wrapper. Normally we require a USAGE privilege (or the like) when creation of one object creates a dependency on another object. See Possible Problem **MED-019**.

General Rules

1) A foreign server descriptor *FSD* is created. *FSD* includes:

   a) The foreign server name *FSN*.

   b) The foreign-data wrapper name *WN*.

   c) The <server type>, if specified.

   d) The <server version>, if specified.

   e) The <authorization identifier>, if specified.

- 1 subrule deleted.

   f) If <generic options> *GO* is specified, then the generic options descriptor created by *GO*; otherwise, an empty generic options descriptor.

2) A privilege descriptor is created that defines the USAGE privilege on this foreign server to the <authorization identifier> of the <foreign server definition>. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <foreign server definition>.

13.2 <alter foreign server statement>

Function

Change the definition of a foreign server.

Format

```
<alter foreign server statement> ::=
       ALTER SERVER <foreign server name>
       [ <new version> ]
       [ <alter generic options> ]

<new version> ::= VERSION <server version>
```

Syntax Rules

1) If <new version> is not specified, then <alter generic options> shall be specified.

2) If <alter generic options> is not specified, then <new version> shall be specified.

3) Let *FSN* be the <foreign server name>. Let *C* be the catalog identified by the explicit or implicit catalog name of *FSN*. *C* shall include a foreign server descriptor *FSD* whose foreign server name is equivalent to *FSN*.

4) If <alter generic options> *AGO* is specified, then the Syntax Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *FSD* as the applicable generic options descriptor.

5) Let *A* be the <authorization identifier> that owns the foreign server identified by *FSN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

****Editor's Note****

The Access Rules for <alter foreign server statement> and <drop foreign server statement> presume that a foreign server has an owner. Is this the same as the <authorization identifier> in <foreign server definition>? This does not appear to be suitable, because the <authorization identifier> is currently optional, and the Description in Definition Schema says that it is the authorization identifier used when connecting to the foreign server, which is a separate notion. See Possible Problem **MED-018**.

General Rules

1) If <new version> *NV* is specified, then the <server version> included in *FSD* is the <server version> specified in *NV*.

2) If <alter generic options> *AGO* is specified, then the General Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *FSD* as the applicable generic options descriptor.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <alter foreign server statement>.

13.3 <drop foreign server statement>

Function

Destroy a foreign server

Format

```
<drop foreign server statement> ::=
       DROP SERVER <foreign server name> <drop behavior>
```

Syntax Rules

1) Let *FSN* be the <foreign server name>. Let *C* be the catalog identified by the explicit or implicit catalog name of *FSN*. *C* shall include a foreign server descriptor *S* whose foreign server name is equivalent to *FSN*.

2) If <drop behavior> specifies RESTRICT, then *S* shall not be referenced by any foreign table descriptor or by any user mapping descriptor.

3) Let *A* be the <authorization identifier> that owns the foreign server identified by *FSN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

1) Let *UM* be any user mapping descriptor that includes a foreign server name that is equivalent to *SN*. Let *AI* be the authorization identifier included in *UM*. The following <drop user mapping statement> is effectively executed without further Access Rule checking:

   DROP USER MAPPING FOR *AI* SERVER *SN*

2) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *SN* FROM *A* CASCADE

3) The descriptor *S* is destroyed.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <drop foreign server statement>.

13.4 <foreign-data wrapper definition>

Function

Define a foreign-data wrapper

Format

```
<foreign-data wrapper definition> ::=
       CREATE FOREIGN DATA WRAPPER <foreign-data wrapper name>
       [ AUTHORIZATION <authorization identifier> ]
       [ <library name specification> ]
       <language clause>
       [ <generic options> ]

<library name specification> ::= LIBRARY <library name>

<library name> ::= <character string literal>
```

Syntax Rules

1) Let *WN* be the <foreign-data wrapper name>. Let *C* be the catalog identified by the explicit or implicit catalog name of *FSN*. *C* shall not include a foreign-data wrapper descriptor whose foreign-data wrapper name is equivalent to *WN*.

2) If <library name specification> is not specified, then a <library name specification> with an implementation-dependent <library name> is implicit.

3) If AUTHORIZATION <authorization identifier> is not specified, then

   Case:

   a) If the <foreign-data wrapper definition> is contained in an SQL-client module that has a <module authorization identifier> specified, then an <authorization identifier> equal to that <module authorization identifier> is implicit for the <foreign-data wrapper definition>.

   b) Otherwise, an <authorization identifier> equal to the SQL-session user identifier is implicit.

Access Rules

1) The privileges necessary to execute <foreign-data wrapper definition> are implementation-defined.

General Rules

1) A foreign-data wrapper descriptor *WD* is created. *WD* includes:

   a) The foreign-data wrapper name *WN*.

   b) The <authorization identifier>.

   c) The name of the library specified in <library name>, if specified.

   d) The name of the language specified in <language clause>.

e) If <generic options> GO is specified, then the generic options descriptor created by *GO*; otherwise, an empty generic options descriptor.

2) A privilege descriptor is created that defines the USAGE privilege on this foreign-data wrapper to the <authorization identifier> of the <foreign-data wrapper definition>. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <foreign-data wrapper definition>.

13.5 <alter foreign-data wrapper statement>

Function

Change the definition of a foreign-data wrapper.

Format

```
<alter foreign-data wrapper statement> ::=
       ALTER FOREIGN DATA WRAPPER <foreign-data wrapper name>
       [ <library name specification> ]
       [ <alter generic options> ]
```

Syntax Rules

1) Let *WN* be the <foreign-data wrapper name>. Let *C* be the catalog identified by the explicit or implicit catalog name of *FSN*. *C* shall include a foreign-data wrapper descriptor *W* whose foreign-data wrapper name is equivalent to *WN*.

2) If <library name specification> is not specified, then <alter generic options> shall be specified.

3) If <alter generic options> is not specified, then <library name specification> shall be specified.

4) If <alter generic options> *AGO* is specified, then the Syntax Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *W* as the applicable generic options descriptor.

5) Let *A* be the <authorization identifier> that owns the foreign-data wrapper identified by *WN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

****Editor's Note****

The Access Rules for <alter foreign-data wrapper statement> and <drop foreign-data wrapper statement> presume that a foreign-data wrapper has an owner, a notion not supported by the descriptor of a foreign-data wrapper, by the <foreign-data wrapper definition> statement, or the Information and Definition Schemas. See Possible Problem **MED-017**.

General Rules

1) If <library name specification> is specified, then the <library name> is included in *W*, replacing any existing <library name>.

2) If <alter generic options> *AGO* is specified, then the General Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *W* as the applicable generic options descriptor.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <alter foreign-data wrapper statement>.

13.6 <drop foreign-data wrapper statement>

Function

Destroy a foreign-data wrapper

Format

```
<drop foreign-data wrapper statement> ::=
       DROP FOREIGN DATA WRAPPER <foreign-data wrapper name> <drop behavior>
```

Syntax Rules

1) Let *WN* be the <foreign-data wrapper name>. Let *C* be the catalog identified by the explicit or implicit catalog name of *FSN*. *C* shall include a foreign-data wrapper descriptor W whose foreign-data wrapper name is equivalent to *WN*.

2) If <drop behavior> specifies RESTRICT, then *W* shall not be referenced by the foreign server name included in any foreign server descriptor.

3) Let *A* be the <authorization identifier> that owns the foreign-data wrapper identified by *WN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

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

   REVOKE ALL PRIVILEGES ON *WN* FROM *A* CASCADE

2) The descriptor of *W* is destroyed.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <drop foreign-data wrapper statement>.

14 Access control

14.1 <revoke statement>

Function

Destroy privileges and role authorizations.

Format

No additional Format items.

Syntax Rules

1) Insert after SR21) Let *T* be any table descriptor included in *S1*. *T* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having USAGE privilege on the foreign server associated with the table described by *T*.

2) Insert after SR36) Let *FS* be any foreign server descriptor. *FS* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having USAGE privilege on the foreign-data wrapper associated with the foreign server described by *FS*.

3) Augment SR37) Add abandoned foreign servers to the list of objects that shall not exist.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR For every abandoned foreign server descriptor *FS*, let *FSN* be the <foreign server name> of *FS*. The following <drop foreign server statement> is effectively executed without further Access Rule checking:

```
DROP SERVER FSN CASCADE
```

Conformance Rules

No additional Conformance Rules.

14.2 <user mapping definition>

Function

Define the mapping of an authorization identifier to a foreign server.

Format

```
<user mapping definition> ::=
       CREATE USER MAPPING FOR <specific or generic authorization identifier>
       SERVER <foreign server name>
       [ <generic options> ]

<specific or generic authorization identifier> ::=
         <authorization identifier>
       | USER
       | CURRENT_USER
```

Syntax Rules

1) Let *FSN* be the <foreign server name> and let *U* be the <specific or generic authorization identifier>. If *U* specifies USER or CURRENT_USER, then the current authorization identifier is implicit.

2) The SQL-environment shall not include an user mapping descriptor whose authorization identifier is *U* and whose foreign server name is equivalent to *FSN*.

Access Rules

None.

****Editor's Note****
Don't we want a rule allowing implementation-defined access rules for creating a user mapping? See Possible Problem **MED-021**.

General Rules

1) An user mapping descriptor *AMD* is created. *AMD* includes:

   a) The authorization identifier *U*.

   b) The foreign server name *FSN*.

   c) If <generic options> *GO* is specified, then the generic options descriptor created by *GO*; otherwise, an empty generic options descriptor.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <user mapping definition>.

14.3 <alter user mapping statement>

Function

Change the definition of an user mapping.

Format

```
<alter user mapping statement> ::=
       ALTER USER MAPPING <specific or generic authorization identifier>
       SERVER <foreign server name>
       <alter generic options>
```

Syntax Rules

1) Let *FSN* be the <foreign server name>, let *AGO* be the <alter generic options> and let *U* be the <specific or generic authorization identifier>. If *U* specifies USER or CURRENT_USER, then the current authorization identifier is implicit.

2) The SQL-environment shall include a user mapping descriptor *AM* whose authorization identifier is *U* and whose foreign server name is equivalent to *FSN*.

3) The Syntax Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *AM* as the applicable generic options descriptor.

Access Rules

None.

General Rules

1) The General Rules of Subclause 11.3, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *AM* as the applicable generic options descriptor.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <alter user mapping statement>.

14.4 <drop user mapping statement>

Function

Destroy an user mapping.

Format

```
<drop user mapping statement> ::=
       DROP USER MAPPING FOR <specific or generic authorization identifier>
       SERVER <foreign server name>
```

Syntax Rules

1) Let *FSN* be the <foreign server name> and let *U* be the <specific or generic authorization identifier>. If *U* specifies USER or CURRENT_USER, then the current authorization identifier is implicit.

2) The SQL-environment shall include a user mapping descriptor *AM* whose authorization identifier is *U* and whose foreign server name is equivalent to *FSN*.

Access Rules

None.

General Rules

1) *AM* is destroyed.

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <drop user mapping statement>.

15 SQL-client modules

15.1 <SQL-client module definition>

Function

Define an SQL-client module.

Format

No additional Format items

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR5)a) If the SQL-session context of any of the SQL-sessions associated with the SQL-agent include {foreign server name : FSConnectionHandle} pairs, then for each such pair:

   i) Let *CH* be the FSConnectionHandle.

- 2 subrules deleted.

   ii) The FreeFSConnection routine is invoked with *CH* as arguments.

2) Insert after GR5)a) If the SQL-session context of any of the SQL-sessions associated with the SQL-agent include {foreign-data wrapper name : WrapperEnvHandle} pairs, then for each such pair:

   i) Let *EH* be the WrapperEnvHandle.

   ii) The FreeWrapperEnv routine is invoked with *EH* as the argument.

Conformance Rules

No additional Conformance Rules.

15.2 Calls to an <externally-invoked procedure>

Function

Define the call to an <externally-invoked procedure> by an SQL-agent.

Syntax Rules

1) Insert into SR2)e)

```
DATA_EXCEPTION_DATALINK_VALUE_EXCEEDS_MAXIMUM_LENGTH:
        constant SQLSTATE_TYPE := "2201D";
DATA_EXCEPTION_INVALID_LINK_TYPE:
        constant SQLSTATE_TYPE := "22016";
DATA_EXCEPTION_INVALID_DATA_SPECIFIED_FOR_DATALINK:
        constant SQLSTATE_TYPE := "22017";
DATA_EXCEPTION_NULL_ARGUMENT_PASSED_TO_DATALINK_CONSTRUCTOR:
        constant SQLSTATE_TYPE := "2201A";
DATALINK_EXCEPTION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "HW000";
DATALINK_EXCEPTION_EXTERNAL_FILE_NOT_LINKED:
        constant SQLSTATE_TYPE := "HW001";
DATALINK_EXCEPTION_EXTERNAL_FILE_ALREADY_LINKED:
        constant SQLSTATE_TYPE := "HW002";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "HV000";

FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_COLUMN_NAME_NOT_FOUND:
        constant SQLSTATE_TYPE := "HV005";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_DYNAMIC_PARAMETER_VALUE_NEEDED:

        constant SQLSTATE_TYPE := "HV002";          FOREIGN_DATA_WRAPPER_SPECIFIC_CON

        constant SQLSTATE_TYPE := "HV003";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_FUNCTION_SEQUENCE_ERROR:
        constant SQLSTATE_TYPE := "HV010";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INCONSISTENT_DESCRIPTOR_INFORMATION:

        constant SQLSTATE_TYPE := "HV021";          FOREIGN_DATA_WRAPPER_SPECIFIC_CON

        constant SQLSTATE_TYPE := "HV024";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_COLUMN_NAME:
        constant SQLSTATE_TYPE := "HV007";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_COLUMN_NUMBER:
        constant SQLSTATE_TYPE := "HV008";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_DATA_TYPE:
        constant SQLSTATE_TYPE := "HV004";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_DESCRIPTOR_FIELD_IDENTIFIER:

        constant SQLSTATE_TYPE := "HV091";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_HANDLE:
        constant SQLSTATE_TYPE := "HV00B";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_OPTION_INDEX:
        constant SQLSTATE_TYPE := "HV00C";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_OPTION_NAME:
        constant SQLSTATE_TYPE := "HV00D";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_STRING_LENGTH_OR_BUFFER_LENGTH:

        constant SQLSTATE_TYPE := "HV090";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_INVALID_USE_OF_NULL_POINTER:
        constant SQLSTATE_TYPE := "HV009";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_LIMIT_ON_NUMBER_OF_HANDLES_EXCEEDED:
```

```
        constant SQLSTATE_TYPE := "HV014";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_MEMORY_ALLOCATION_ERROR:
        constant SQLSTATE_TYPE := "HV001";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_OPTION_NAME_NOT_FOUND:
        constant SQLSTATE_TYPE := "HV00J";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_REPLY_HANDLE:
        constant SQLSTATE_TYPE := "HY00K";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_UNABLE_TO_CREATE_EXECUTION:
        constant SQLSTATE_TYPE := "HV00L";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_UNABLE_TO_CREATE_REPLY:
        constant SQLSTATE_TYPE := "HV00M";
FOREIGN_DATA_WRAPPER_SPECIFIC_CONDITION_UNABLE_TO_ESTABLISH_CONNECTION_TO_FOREIGN_SER

        constant SQLSTATE_TYPE := "HV00N";
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

15.3 <SQL procedure statement>

Function

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

Format

```
<SQL schema definition statement> ::=
         !! All alternatives from ISO/IEC 9075-2
       | !! All alternatives from ISO/IEC 9075-5
       | <foreign table definition>
       | <foreign server definition>
       | <foreign-data wrapper definition>
       | <user mapping definition>

<SQL schema manipulation statement> ::=
         !! All alternatives from ISO/IEC 9075-2
       | !! All alternatives from ISO/IEC 9075-5
       | <alter foreign table statement>
       | <drop foreign table statement>
       | <alter foreign server statement>
       | <drop foreign server statement>
       | <alter foreign-data wrapper statement>
       | <drop foreign-data wrapper statement>
       | <alter user mapping statement>
       | <drop user mapping statement>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

15.4 Data type correspondences

Function

Specify the data type correspondences for SQL data types and host language types.

Tables

Table 6—Data type correspondences for Ada

SQL Data Type	Ada Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	
DATALINK	SQL_STANDARD.CHAR, with P'LENGTH of *LD*[1]

[1]The length *LD* of the character data type corresponding with SQL data type DATALINK is the smallest integer not less than the quotient of the division *N*/*B*, where *N* is the maximum datalink length and *B* is the implementation-defined number of octets contained in a character of the host language. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)

Table 7—Data type correspondences for C

SQL Data Type	C Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	
DATALINK	char, with length *LD*[5]

[5]The length *LD* of the character data type corresponding with SQL data type DATALINK is the smallest integer not less than the quotient of the division *N*/*B*, where *N* is the maximum datalink length and *B* is the implementation-defined number of octets contained in a character of the host language. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)

Table 8—Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	
DATALINK	alphanumeric, with length *LD*[4]

[4]The length *LD* of the character data type corresponding with SQL data type DATALINK is the smallest integer not less than the quotient of the division *N*/*B*, where *N* is the maximum datalink length and *B* is the implementation-defined number of octets contained in a character of the host language. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)

Table 9—Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	
DATALINK	CHARACTER with length *LD*[4]

[4]The length *LD* of the character data type corresponding with SQL data type DATALINK is the smallest integer not less than the quotient of the division *N*/*B*, where *N* is the maximum datalink length and *B* is the implementation-defined number of octets contained in a character of the host language. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)

Table 10—Data type correspondences for MUMPS

SQL Data Type	MUMPS Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	
DATALINK	character

Table 11—Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	
DATALINK	PACKED ARRAY[1..*LD*[2]] OF CHAR

[2]The length *LD* of the character data type corresponding with SQL data type DATALINK is the smallest integer not less than the quotient of the division *N*/*B*, where *N* is the maximum datalink length and *B* is the implementation-defined number of octets contained in a character of the host language. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)

Table 12—Data type correspondences for PL/I

SQL Data Type	PL/I Data Type
All alternatives from ISO/IEC 9075-2	
All alternatives from ISO/IEC 9075-5	

Table 12—Data type correspondences for PL/I (Cont.)

SQL Data Type	PL/I Data Type
DATALINK	CHARACTER VARYING(*LD*[2])
[2]The length *LD* of the character data type corresponding with SQL data type DATALINK is the smallest integer not less than the quotient of the division *N*/*B*, where *N* is the maximum datalink length and *B* is the implementation-defined number of octets contained in a character of the host language. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)	

Conformance Rules

No additional Conformance Rules.

16 Data manipulation

16.1 Effect of deleting rows from base tables

Function

Specify the effect of deleting rows from one or more base tables.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR6) For each row *R* that is marked for deletion from *T*, for each column *DLC* of *T* whose declared type is DATALINK or some distinct type with a source data type of DATALINK and whose descriptor includes a <datalink control definition> that specifies FILE LINK CONTROL, let *DLCV* be the value of *DLC* in *R*.

2) Insert after GR6) If *DLCV* is not the null value, then *EF* be the external file referenced by *DLCV*.

Case:

a) If *EF* is linked, then *EF* is unlinked.

NOTE 14 – The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in Subclause 4.9, "Columns, fields, and attributes".

b) Otherwise, an exception condition is raised: *datalink exception — external file not linked*.

Conformance Rules

No additional Conformance Rules.

16.2 Effect of inserting tables into base tables

Function

Specify the effect of inserting each of one or more given tables into its associated base table.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR For each row *R* inserted into *T*, for each column *DLC* of *T* whose declared type is DATALINK or some distinct type with a source data type of DATALINK and whose descriptor includes a <datalink control definition> that specifies FILE LINK CONTROL and includes a <datalink file control option> that is INTEGRITY ALL or INTEGRITY SELECTIVE, let *DLCV* be the value of *DLC* in *R*.

2) Insert this GR If *DLCV* is not the null value, then let *EF* be the external file referenced by *DLCV*.

Case:

a) If *EF* is linked, then an exception condition is raised: *datalink exception — external file already linked*.

b) If INTEGRITY ALL is specified, then *EF* is linked according to the <datalink file control option> READ PERMISSION and WRITE PERMISSION of *DLC*.

c) If INTEGRITY SELECTIVE is specified, then *EF* may be linked in an implementation-defined manner according to the <datalink file control option> READ PERMISSION and WRITE PERMISSION of *DLC*.

Conformance Rules

No additional Conformance Rules.

16.3 Effect of replacing rows in base tables

Function

Specify the effect of replacing some of the rows in one or more base tables.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR7) For each replaced row *R*, for each column *DLC* of *R* whose declared type is DATALINK or some distinct type with a source data type of DATALINK and whose descriptor includes a <datalink control definition> that specifies FILE LINK CONTROL, let *DLCV1* be the existing value of *DLC* in *R* and let *DLCV2* be the correponding datalink in the new transition variable.

2) Insert after GR7) If *DLCV1* is not the null value, then let *EF1* be the external file referenced by *DLCV1*.

Case:

a) If *EF1* is not linked, then an exception condition is raised: *datalink exception — external file not linked*.

- 1 subrule deleted.

b) Otherwise, *EF1* is unlinked.

NOTE 15 – The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in Subclause 4.9, "Columns, fields, and attributes".

3) Insert after GR7) If *DLCV2* is not the null value, then let *EF2* be the external file referenced by *DLCV2*.

Case:

a) If *EF2* is linked, then an exception condition is raised: *datalink exception — external file already linked*.

b) Otherwise, *EF2* is linked according to the <datalink file control option> of READ PERMISSION and WRITE PERMISSION of *DLC*.

Conformance Rules

No additional Conformance Rules.

17 Dynamic SQL

17.1 Description of SQL descriptor areas

Function

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

Syntax Rules

1) Insert before SR6)q) TYPE indicates DATALINK.

2) Insert before SR7)v) TYPE indicates DATALINK and *T* is specified by DATALINK.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR1) Table 13, "Codes used for SQL data types in Dynamic SQL", specifies the codes associated with the SQL data types.

Table 13—Codes used for SQL data types in Dynamic SQL

Data Type	Code
All alternatives from ISO/IEC 9075-5	*All alternatives from ISO/IEC 9075-5*
DATALINK	50

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", TYPE shall not indicate DATALINK.

17.2 <describe statement>

Function

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

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR8)d)xi) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum datalink length.

NOTE 16 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", TYPE shall not indicate DATALINK.

18 Embedded SQL

18.1 <embedded SQL Ada program>

Function

Specify an <embedded SQL Ada program>.

Format

```
<Ada derived type specification> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <Ada DATALINK variable>

<Ada DATALINK variable> ::=
       SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR5)i) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   shall be replaced by

   ```
   Interfaces.SQL.CHAR(1..MDL)
   ```

   where *MDL* is the maximum datalink length, in any <Ada DATALINK variable>.

   NOTE 17 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <Ada DATALINK variable>.

18.2 <embedded SQL C program>

Function

Specify an <embedded SQL C program>.

Format

```
<C derived variable> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <C DATALINK variable>

<C DATALINK variable> ::=
       SQL TYPE IS <datalink type>
         <C host identifier> [ <C initial value> ]
           [ <comma> <C host identifier> [ <C initial value> ] }... ]
```

Syntax Rules

1) Insert after SR6)n) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   shall be replaced by

   ```
   char[MDL]
   ```

   where *MDL* is the maximum datalink length, in any <C DATALINK variable>.

   NOTE 18 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <C DATALINK variable>.

18.3 <embedded SQL COBOL program>

Function

Specify an <embedded SQL COBOL program>.

Format

```
<COBOL derived type specification> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <COBOL DATALINK variable>

<COBOL DATALINK variable> ::=
       [ USAGE [ IS ] ]
       SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR6)k) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   shall be replaced by

   ```
   PIC X(MDL).
   ```

   where *MDL* is the maximum datalink length, in any <COBOL DATALINK variable>.

   NOTE 19 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <COBOL DATALINK variable>.

18.4 <embedded SQL Fortran program>

Function

Specify an <embedded SQL Fortran program>.

Format

```
<Fortran derived type specification> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <Fortran DATALINK variable>

<Fortran DATALINK variable> ::=
       SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR6)k) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   for a given <Fortran host identifier> *fhi* shall be replaced by

   ```
   CHARACTER fhi * MDL
   ```

   where *MDL* is the maximum datalink length, in any <Fortran DATALINK variable>.

   NOTE 20 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <Fortran DATALINK variable>.

18.5 <embedded SQL MUMPS program>

Function

Specify an <embedded SQL MUMPS program>.

Format

```
<MUMPS derived type specification> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <MUMPS DATALINK variable>

<MUMPS DATALINK variable> ::=
       SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR9)h) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   for a given <MUMPS host identifier> *mhi* shall be replaced by

   ```
   VARCHAR mhi MDL
   ```

   where *MDL* is the maximum datalink length, in any <MUMPS DATALINK variable>.

   NOTE 21 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <MUMPS DATALINK variable>.

18.6 <embedded SQL Pascal program>

Function

Specify an <embedded SQL Pascal program>.

Format

```
<Pascal derived type specification> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <Pascal DATALINK variable>

<Pascal DATALINK variable> ::=
       SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR5)l) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   shall be replaced by

   ```
   PACKED ARRAY [1..MDL] OF CHAR
   ```

   where *MDL* is the maximum datlaink length, in any <Pascal DATALINK variable>.

   NOTE 22 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <Pascal DATALINK variable>.

18.7 <embedded SQL PL/I program>

Function

Specify an <embedded SQL PL/I program>.

Format

```
<PL/I derived type specification> ::=
       | !! All alternatives from ISO/IEC 9075-5
       | <PL/I DATALINK variable>

<PL/I DATALINK variable> ::=
       SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR5)i) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   shall be replaced by

   ```
   CHARACTER(MDL)
   ```

   where *MDL* is the maximum datalink length, in any <PL/I DATALINK variable>.

   NOTE 23 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M004, "Datalinks via SQL language", conforming SQL language shall not specify <PL/I DATALINK variable>.

19 Call-Level Interface specifications

19.1 <CLI routine>

Format

Describe a generic SQL/CLI routine.

Format

```
<CLI routine> ::=
       | !! All alternatives from ISO/IEC 9075-3
       | BuildDataLink
       | GetDataLinkAttr
```

Syntax Rules

Table 14—Abbreviated SQL/CLI generic names

Generic Name	Abbreviation
	All alternatives from ISO/IEC 9075-3
BuildDataLink	BDL
GetDataLinkAttr	GDL

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

19.2 Implicit DESCRIBE USING clause

Function

Specify the rules for an implicit DESCRIBE USING clause.

General Rules

1) Insert after GR5)c)iv)11) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.

2) Insert after GR8)d)vi)11) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.

19.2.1 CLI-specific status codes

Table 15—SQLSTATE class and subclass values for SQL/CLI-specific conditions

Category	Condition	Class	Subcondition	Subclass
	All alternatives from ISO/IEC 9075-3			
X	CLI-specific condition	HY	invalid datalink value	093

19.2.2 Description of CLI item descriptor areas

Function

Specify the identifiers, data types and codes for fields used in CLI item descriptor areas.

Syntax Rules

1) Insert after SR5)c)xiv) TYPE indicates DATALINK.

2) Replaces SR7)c)iv) TYPE indicates DEFAULT, CHARACTER, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY LARGE OBJECT LOCATOR, USER-DEFINED TYPE LOCATOR, DATALINK, REF, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

3) Insert after SR12)c)ix) TYPE indicates DATALINK and one of the following is true:

   1) NULL is true.

   2) DEFERRED is true.

4) Replace SR13)c)iv) TYPE indicates DEFAULT, CHARACTER, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY LARGE OBJECT LOCATOR, USER-DEFINED TYPE LOCATOR, DATALINK, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

General Rules

Table 16—Codes used for implementation data types in SQL/CLI

Data Type	Code
	All alternatives from ISO/IEC 9075-3
DATALINK	50

Table 17—Codes used for application data types in SQL/CLI

Data Type	Code
	All alternatives from ISO/IEC 9075-3
DATALINK	50

19.2.3 Other tables associated with CLI

Table 18—Codes used to identify SQL/CLI routines

Generic Name	Code
	All alternatives from ISO/IEC 9075-3
BuildDataLink	1029
GetDataLinkAttr	1034

Table 19—Codes and data types for implementation information

Information Type	Code	Data Type
All alternatives from ISO/IEC 9075-3		
MAXIMUM DATALINK LENGTH	20004	INTEGER

Table 20—Codes used for datalink attributes

Attribute	Code
COMMENT	1
TYPE	2
URL COMPLETE	3
URL PATH	4
URL PATH ONLY	5
URL SCHEME	6
URL SERVER	7
Implementation-defined datalink attribute	<0

Table 21—Data types of attributes

Attribute	Data type	Values
	All alternatives from ISO/IEC 9075-3	
Datalink attributes		
COMMENT	CHARACTER VARYING(*L*)[1]	Datalink comment value
TYPE	CHARACTER VARYING(*L*)[1]	Datalink type value
URL COMPLETE	CHARACTER VARYING(*L*)[1]	Datalink complete URL
URL PATH	CHARACTER VARYING(*L*)[1]	Datalink URL path
URL PATH ONLY	CHARACTER VARYING(*L*)[1]	Datalink URL path only
URL SCHEME	CHARACTER VARYING(*L*)[1]	Datalink URL schema
URL SERVER	CHARACTER VARYING(*L*)[1]	Datalink URL server
Implementation-defined datalink attribute	Implementation-defined data type	Implementation-defined value

[1]Where *L* is an implementation-defined integer not less than the maximum datalink length. (The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".)

Conformance Rules

1) Without Feature M003, "Datalinks via SQL/CLI", conforming SQL language shall not specify `BuildDataLink( )`.

2) Without Feature M003, "Datalinks via SQL/CLI", conforming SQL language shall not specify `GetDataLinkAttr( )`.

19.3 SQL/CLI data type correspondences

Function

Replaces first paragraph Specify the SQL/CLI data type correspondences for SQL data types and host language types associated with the required parameter mechanisms, as shown in Table 3, "Supported calling conventions of SQl/CLI routines by language", in ISO/IEC 9075-3.

Tables

Table 22—SQL/CLI data type correspondences for Ada

SQL Data Type	Ada Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

Table 23—SQL/CLI data type correspondences for C

SQL Data Type	C Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

Table 24—SQL/CLI data type correspondences for COBOL

SQL Data Type	COBOL Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

Table 25—SQL/CLI data type correspondences for Fortran

SQL Data Type	Fortran Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

Table 26—SQL/CLI data type correspondences for MUMPS

SQL Data Type	MUMPS Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

Table 27—SQL/CLI data type correspondences for Pascal

SQL Data Type	Pascal Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

Table 28—SQL/CLI data type correspondences for PL/I

SQL Data Type	PL/I Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*

20 SQL/CLI routines

20.1 BuildDataLink

Function

Build a datalink value.

Definition

```
BuildDataLink (
      StatementHandle       IN  INTEGER,
      LinkType              IN  CHARACTER(L1),
      LinkTypeLength        IN  INTEGER,
      DataLocation          IN  CHARACTER(L2),
      DataLocationLength    IN  INTEGER,
      Comment               IN  CHARACTER(L3),
      CommentLength         IN  INTEGER,
      DataLink              OUT CHARACTER(L4),
      BufferLength          IN  INTEGER,
      StringLength          OUT INTEGER )
      RETURNS SMALLINT
```

where *L1*, *L2*, and *L3* are determined by the values of LinkTypeLength, DataLocationLength, and CommentLength and *L4* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a datalink.

General Rules

1) Let *SH* be the value of StatementHandle.

   NOTE 24 – *SH* is used only if BuildDataLink issues a completion or exception condition.

2) Let *DL* be the datalink value whose *LinkType* is LinkType, whose *Scheme*, *File Server*, and *File Path* are DataLocation, and whose *Comment* is Comment.

   NOTE 25 – *LinkType*, *Scheme*, *File Server*, *File Path*, and *Comment* are defined in Subclause 4.7, "Datalinks".

3) Let *DLL* be the length in octets of *DL*.

4) If *DLL* is greater than the maximum datalink length, then an exception condition is raised: *CLI-specific condition — invalid datalink value.*

   NOTE 26 – The term "maximum datalink length" is defined in Subclause 4.7, "Datalinks".

5) Apply the General Rules of Subclause 5.9, "Character string retrieval", in ISO/IEC 9075-3 with DataLink, *DL*, BufferLength, and StringLength as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M003, "Datalinks via SQL/CLI", conforming SQL language shall not specify `BuildDataLink( )`.

20.2 GetDataLinkAttr

Function

Retrieve the value of a datalink attribute.

Definition

```
GetDataLinkAttr (
      StatementHandle       IN  INTEGER,
      Attribute             IN  SMALLINT,
      DataLink              IN  CHARACTER(L),
      DataLinkLength        IN  INTEGER,
      Value                 OUT ANY,
      BufferLength          IN  INTEGER,
      StringLength          OUT INTEGER )
      RETURNS SMALLINT
```

where *L* is determined by the value of DataLinkLength.

General Rules

1) Let *SH* be the value of StatementHandle.
   NOTE 27 – *SH* is used only if GetDataLinkAttr issues a completion or exception condition.

2) Let *A* be the value of Attribute.

3) If *A* is not one of the code values in Table 20, "Codes used for datalink attributes", then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

4) Let *DLL* be the value of DataLinkLength.

5) Case:

   a) If *DLL* is not negative, then let *L* be *DLL*.

   b) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

6) Case:

   a) If *L* is zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *DL* be the first *L* octets of DataLink.

7) Case:

   a) If *L* is zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of DataLink and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *invalid cursor name*.

ii) Otherwise, let *DL* be the first *L* octets of Datalink.

8) Let *ML* be the implementation-defined maximum length in characters of a datalink value.

9) If *DL* is not a valid datalink value, then an exception condition is raised: *CLI-specific condition — invalid datalink value.*

10) Let *BL* be the value of BufferLength.

11) If *A* specifies an implementation-defined datalink attribute, then

Case:

a) If the data type for the datalink attribute is specified as INTEGER in Table 21, "Data types of attributes", then Value is set to the value of the implementation-defined datalink attribute and no further General Rules of this Subclause are applied.

b) Otherwise:

i) Let *AV* be the value of the implementation-defined datalink attribute.

ii) The General Rules of Subclause 5.9, "Character string retrieval", in ISO/IEC 9075-3 are applied with Value, *AV*, *BL*, and StringLength as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

12) Case:

a) If *A* indicates COMMENT, then *AV* is set to the value of the datalink comment of *DL*.

b) If *A* indicates TYPE, then *AV* is set to the value of datalink type of *DL*.

c) If *A* indicates URL COMPLETE, then *AV* is set to the value of the concatenation of the datalink schema, file server name, and file path of *DL*.

d) If *A* indicates URL PATH, then *AV* is set to the value of the concatenation of the datalink operating system path, external file name of *DL*, possibly combined with an access token under the General Rules of Subclause 6.3, "<string value function>".

e) If *A* indicates URL PATH ONLY, then *AV* is set to the value of the concatenation of the datalink operating system path and external file name of *DL*.

f) If *A* indicates URL SCHEME, then *AV* is set to the value of the datalink scheme of *DL*.

g) If *A* indicates URL SERVER, then *AV* is set to the value of the datalink external file server of *DL*.

NOTE 28 – The datalink attributes are defined in Subclause 4.7, "Datalinks".

13) The General Rules of Subclause 5.9, "Character string retrieval", in ISO/IEC 9075-3 are applied with Value, *AV*, *BL*, and StringLength as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M003, "Datalinks via SQL/CLI", conforming SQL language shall not specify `GetDataLinkAttr( )`.

20.3 GetInfo

Function

Get information about the implementation.

Definition

No additional Definition items.

General Rules

1) Insert into the dashed list in GR10)a)

   — MAXIMUM DATALINK LENGTH

Conformance Rules

1) Without Feature M003, "Datalinks via SQL/CLI", the value of InfoType shall not indicate MAXIMUM DATALINK LENGTH.

21 SQL/MED common specifications

21.1 Description of MED item descriptor areas

Function

Specify the identifiers, data types and codes for fields used in MED item descriptor areas.

Syntax Rules

1) An SQL/MED item descriptor area comprises the fields specified in Table 5, "Fields in SQL/MED descriptor areas".

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

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

   Case:

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

   b) If *DT* is ARRAY, then the descriptor of the associated element type of *DT*. The subordinate descriptors of *DE* are those descriptors that are immediately subordinate descriptors of *DE* or that are subordinate descriptors of a descriptor that is immediately subordinate to *DE*.

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

   a) SDE_1 is in the first ordinal position.

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

5) Let *IDA* be an item descriptor area in an wrapper parameter descriptor. *IDA* is *valid* if and only if all of the following are true:

   a) TYPE is one of the code values in Table 16, "Codes used for implementation data types in SQL/CLI".

   b) If LEVEL is 0 (zero) for *IDA*, then let *TLC* be the value of TOP_LEVEL_COUNT of the wrapper parameter descriptor associated with *IDA*. *IDA* shall be one of exactly *TLC* item descriptor areas in the wrapper parameter descriptor.

c) Exactly one of the following is true:

Case:

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

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

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

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

v) TYPE indicates CHARACTER or CHARACTER VARYING, or CHARACTER LARGE OBJECT, and LENGTH is a valid length value for a <character string type>.

vi) TYPE indicates BIT or BIT VARYING, and LENGTH is a valid length value for <bit string type>.

vii) TYPE indicates BINARY LARGE OBJECT, and LENGTH is a valid value for a <binary string type>.

viii) TYPE indicates a <datetime type>, DATETIME_INTERVAL_CODE is one of the code values in Table 9, "Codes associated with datetime data types in SQL/CLI", in ISO/IEC 9075-3, and PRECISION is a valid value for the <time precision> or <timestamp precision> of the indicated datetime data type.

ix) TYPE indicates an <interval type>, DATETIME_INTERVAL_CODE is one of the code values in Table 10, "Codes associated with <interval qualifier> in SQL/CLI", in ISO/IEC 9075-3, to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is a valid <interval leading field precision>, and PRECISION is a valid value for <interval fractional seconds precision>, if applicable.

x) TYPE indicates REF.

xi) TYPE indicates USER-DEFINED TYPE.

xii) TYPE indicates BOOLEAN.

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

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

****Editor's Note****

What does the phrase "there is LOCATOR" mean? In fact, the entire subrule above doesn't seem to make sense.

xv) TYPE indicates an implementation-defined data type.

6) Let *HL* be the standard programming language of the invoking SQL-server. Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 5.15, "SQL/CLI data type correspondences", ISO/IEC 9075-3. Refer to the two columns of the operative data type correspondence table as the "SQL data type column" and the "host data type column".

7) An SQL/MED item descriptor area in an SQL/MED descriptor area that is not a wrapper row descriptor or a table reference descriptor is *consistent* if and only if all of the following are true:

   a) TYPE indicates DEFAULT or is one of the code values in Table 17, "Codes used for application data types in SQL/CLI".

   b) All of the following are true:

      i) TYPE is one of the code values in Table 17, "Codes used for application data types in SQL/CLI".

      ii) TYPE is neither ROW nor ARRAY.

      iii) The row that contains the SQL data type corresponding to TYPE in the SQL data type column of the operative data type correspondence table does not contain "None" in the host data type column.

   c) Exactly one of the following is true:

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

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

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

      iv) TYPE indicates DEFAULT, CHARACTER, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY LARGE OBJECT LOCATOR, USER-DEFINED TYPE LOCATOR, REF, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

      v) TYPE indicates ROW in a wrapper row descriptor and, where *N* is the value of the DEGREE field in the corresponding item descriptor area in the wrapper parameter descriptor, there are exactly *N* immediately subordinate descriptor areas of *IDA*, and those item descriptor areas are valid.

      vi) TYPE indicates ARRAY or ARRAY LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that item descriptor area is valid.

      vii) TYPE indicates an implementation-defined data type.

8) Let *IDA* be an SQL/MED item descriptor area in a server parameter descriptor. Let *IDA1* be the corresponding item descriptor area in the wrapper parameter descriptor.

9) If the OCTET_LENGTH_POINTER field of *IDA* has the same non-zero value as the INDICATOR_POINTER field of *IDA*, then SHARE is true for *IDA*; otherwise SHARE is false for *IDA*.

10) Case:

    a) If SHARE is true and the value of the commonly addressed host variable is the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in CLI", in ISO/IEC 9075-3, then NULL is true for *IDA*.

    b) If SHARE is false, INDICATOR_POINTER is not zero, and the value of the host variable addressed by INDICATOR_POINTER is the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in CLI", in ISO/IEC 9075-3, then NULL is true for *IDA*.

    c) Otherwise, NULL is false for *IDA*.

11) If NULL is false, OCTET_LENGTH_POINTER is not zero, and the value of the host variable addressed by OCTET_LENGTH_POINTER the appropriate 'Code' for DATA AT EXEC in Table 26, "Miscellaneous codes used in CLI", in ISO/IEC 9075-3, then DEFERRED is true for *IDA*; otherwise DEFERRED is false for *IDA*.

12) *IDA* is *valid* if and only if:

    a) TYPE is one of the code values in Table 17, "Codes used for application data types in SQL/CLI", and at least one of the following is true:

        i) TYPE is ROW or ARRAY.

        ii) The row of the operative data type correspondences table that contains the SQL data type corresponding to the value of TYPE in the SQL data type column does not contain 'None' in the host data type column.

    b) If LEVEL is 0 (zero) for *IDA*, then let *TLC* be the value of TOP_LEVEL_COUNT in the server parameter descriptor associated with *IDA*. *IDA* shall be one of exactly *TLC* item descriptor areas in the wrapper parameter descriptor.

    c) One of the following is true:

        Case:

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

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

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

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

        v) TYPE indicates CHARACTER, CHARACTER LARGE OBJECT, or BINARY LARGE OBJECT, and one of the following is true:

            1) NULL is true.

            2) DEFERRED is true.

3) OCTET_LENGTH_POINTER is not zero, PARAMETER_MODE in *IDA1* is PARAM MODE IN or PARAM MODE INOUT, the value *V* of the host variable addressed by OCTET_LENGTH_POINTER is greater than zero, and the number of characters wholly contained in the first *V* octets of the host variable addressed by DATA_POINTER is a valid length value for a CHARACTER, CHARACTER LARGE OBJECT, or BINARY LARGE OBJECT data type, as indicated by TYPE.

4) OCTET_LENGTH_POINTER is not zero, PARAMETER_MODE in *IDA1* is PARAM MODE IN or PARAM MODE INOUT, the value of the host variable addressed by OCTET_LENGTH_POINTER indicates NULL TERMINATED, and the number of characters of the value of the host variable addressed by DATA_POINTER that precede the implementation-defined null character that terminates a C character string is a valid length value for a CHARACTER, CHARACTER LARGE OBJECT, or BINARY LARGE OBJECT data type, as indicated by TYPE.

5) OCTET_LENGTH_POINTER is zero, PARAMETER_MODE in *IDA1* is PARAM MODE IN or PARAM MODE INOUT, and the number of characters of the value of the host variable addressed by DATA_POINTER that precede the implementation-defined null character that terminates a C character string is a valid length value for a CHARACTER, CHARACTER LARGE OBJECT, or BINARY LARGE OBJECT data type, as indicated by TYPE.

6) PARAMETER_MODE in *IDA1* is PARAM MODE OUT.

vi) TYPE indicates REF and one of the following is true:

1) NULL is true.

2) DEFERRED is true.

vii) TYPE indicates CHARACTER LARGE OBJECT LOCATOR, BINARY LARGE OBJECT LOCATOR, or USER-DEFINED TYPE LOCATOR and one of the following is true:

1) NULL is true.

2) DEFERRED is true.

viii) TYPE indicates ROW and, where *N* is the value of the DEGREE field in the corresponding item descriptor area in the wrapper parameter descriptor, there are exactly *N* immediately subordinate descriptor areas of *IDA*, and those item descriptor areas are valid.

ix) TYPE indicates ARRAY or ARRAY LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that item descriptor area is valid.

x) TYPE indicates an implementation-defined data type.

d) One of the following is true:

i) DATA_POINTER is zero and NULL is true.

ii) DATA_POINTER is zero and DEFERRED is true.

iii) DATA_POINTER is not zero and exactly one of the following is true:

1) NULL is true.

2) DEFERRED is true.

3) PARAMETER_MODE in *IDA1* is PARAM MODE IN or PARAM MODE INOUT and the value of the host variable addressed by DATA_POINTER is a valid value of the data type indicated by TYPE.

4) PARAMETER_MODE in *IDA1* is PARAM MODE OUT.

13) An SQL/MED item descriptor area in an server row descriptor is valid if and only if:

a) TYPE is one of the code values in Table 17, "Codes used for application data types in SQL/CLI", and at least one of the following is true:

i) TYPE is ROW or ARRAY.

ii) The row of the operative data type correspondences table that contains the SQL data type corresponding to the value of TYPE in the SQL data type column does not contain 'None' in the host data type column.

b) If LEVEL is 0 (zero) for IDA, then let *TLC* be the value of TOP_LEVEL_COUNT in the server parameter descriptor associated with *IDA*. *IDA* shall be one of exactly *TLC* item descriptor areas in the wrapper parameter descriptor.

c) One of the following is true:

Case:

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

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

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

iv) TYPE indicates CHARACTER, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY LARGE OBJECT LOCATOR, USER-DEFINED TYPE LOCATOR, REF, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

v) TYPE indicates ROW and, where *N* is the value of the DEGREE field in the corresponding item descriptor area in the wrapper parameter descriptor, there are exactly *N* immediately subordinate descriptor areas of *IDA*, and those item descriptor areas are valid.

vi) TYPE indicates ARRAY or ARRAY LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that item descriptor area is valid.

vii) TYPE indicates an implementation-defined data type.

General Rules

None.

Conformance Rules

None.

21.2 Implicit cursor

Function

Specify the rules for an implicit DECLARE CURSOR and OPEN statement.

General Rules

1) Let *SS* and *AS* be a *SELECT SOURCE* and *ALLOCATED STATEMENT* specified in an application of this Subclause.

2) If there is no cursor associated with *AS*, then a cursor is associated with *AS* and the cursor name associated with *AS* becomes the name of the cursor.

3) The General Rules of Subclause 21.5, "Implicit EXECUTE USING and OPEN USING clauses", are applied to 'OPEN', *SS*, and *AS* as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

4) Let *CT* be 'NO SCROLL'.

5) Let *CS* be 'ASENSITIVE'.

6) Let *CH* be 'WITHOUT HOLD'.

7) Let *CN* be the name of the cursor associated with *AS* and let *CR* be the following <declare cursor>:

```
DECLARE CN CS CT CURSOR CH FOR SS
```

8) Cursor *CN* is opened in the following steps:

   a) A copy of *SS* is effectively created in which:

      i) Each <dynamic parameter specification> is replaced by the value of the corresponding dynamic parameter.

      ii) Each <value specification> generally contained in *SS* that is CURRENT_USER, CURRENT_ROLE, SESSION_USER, or SYSTEM_USER is replaced by the value resulting from evaluation of CURRENT_USER, CURRENT_ROLE, SESSION_USER, or SYSTEM_USER, respectively, with all such evaluations effectively done at the same instant in time.

      iii) Each <datetime value function> generally contained in *SS* is replaced by the value resulting from evaluation of that <datetime value function>, with all such evaluations effectively done at the same instant in time.

      iv) 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.

   b) Let *T* be the table specified by the copy of *SS*.

   c) A table descriptor for *T* is effectively created.

d) The General Rules of Subclause 14.1, "<declare cursor>", in ISO/IEC 9075-2, are applied to *CR*.

e) Cursor *CN* is placed in the open state and its position is before the first row of *T*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

21.3 Implicit DESCRIBE INPUT USING clause

Function

Populate a specified descriptor area with information about the input values required to execute a specified SQL statement.

General Rules

1) Let *S* and *DESC* be a *SOURCE* and a *DESCRIPTOR* specified in the rules of this Subclause.

2) Let *HL* be the standard programming language of the invoking SQL-server.

3) The value of DYNAMIC_FUNCTION and DYNAMIC_FUNCTION_CODE in *DESC* are respectively a character string representation of the prepared statement and a numeric code that identifies the prepared statement.

4) If POPULATE WPD is *false* , then no further rules of this Subclause are applied.

5) If POPULATE WPD is *true* , then a descriptor for the <dynamic parameter specification>s for the prepared statement is stored in *DESC* as follows:

   a) Let *D* be the number of <dynamic parameter specification>s in *S*.

      Case:

      i) If the value of the statement attribute NEST DESCRIPTOR is *true* , then let NS_i, 1 (one) $\leq i \leq D$, be the number of subordinate descriptors of the descriptor for the *i*-th input dynamic parameter.

      ii) Otherwise, let NS_i, 1 (one) $\leq i \leq D$, be 0 (zero).

   b) TOP_LEVEL_COUNT is set to *D*. If *D* is 0 (zero), then let *TD* be 0 (zero); otherwise, let *TD* be $D + \sum_{i=1}^{D} (NS_i)$. COUNT is set to *TD*.
      NOTE 29 – The KEY_TYPE field is not relevant in this case.

   c) If *TD* is zero, then no item descriptor areas are set. Otherwise, the first *TD* item descriptor areas are set so that the *i*-th item descriptor area contains a descriptor of the *j*-th <dynamic parameter specification> such that:

      i) The descriptor for the first such <dynamic parameter specification> is assigned to the first descriptor area.

      ii) The descriptor for the *j*+1-th <dynamic parameter specification> is assigned to the $i+NS_j$+1-th item descriptor area.

      iii) If the value of the statement attribute NEST DESCRIPTOR is *true* , then the implicitly ordered subordinate descriptors for the *j*-th <dynamic parameter specification> are assigned to contiguous item descriptor areas starting at the *i*+1-th item descriptor area.

   d) The descriptor of a <dynamic parameter specification> consists of values for LEVEL, TYPE, NULLABLE, NAME, UNNAMED, PARAMETER_MODE, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, PARAMETER_SPECIFIC_NAME, and other fields depending on the value of TYPE as described below. Those fields and fields that are not applicable for a particular value of

TYPE are set to implementation-dependent values. The DATA_POINTER, INDICATOR_POINTER, OCTET_LENGTH_POINTER, RETURNED_CARDINALITY_POINTER, and KEY_MEMBER fields are not relevant in this case.

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

ii) TYPE is set to a code as shown in Table 7, "Codes used for implementation data types in SQL/CLI", in ISO/IEC 9075-3, indicating the data type of the <dynamic parameter specification> or subordinate descriptor.

iii) NULLABLE is set to 1 (one).
NOTE 30 – This indicates that the <dynamic parameter specification> can have the null value.

iv) KEY_MEMBER is set to 0 (zero).

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

vi) Case:

1) If TYPE indicates a <character string type>, then: LENGTH is set to the length or maximum length in characters of the character string and OCTET_LENGTH is set the maximum possible length in octets of the character string; if *HL* is C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string's collation.

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

3) If TYPE indicates a <binary string type>, then LENGTH is set to the maximum length in octets of the binary string.

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

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

6) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 9, "Codes associated with datetime data types in SQL/CLI", in ISO/IEC 9075-3, to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision> if either is applicable.

7) If TYPE indicates INTERVAL, then LENGTH is set to the length in positions of the interval type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 10, "Codes associated with <interval qualifier> in SQL/CLI", in ISO/IEC 9075-3, to indicate the specific <interval qualifier>, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and PRECISION is set to the <interval fractional seconds precision>, if applicable.

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

9) If TYPE indicates USER-DEFINED TYPE, then USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are set to the <user-defined type name> of the user-defined type. SPECIFIC_TYPE_CATALOG, SPECIFIC_TYPE_SCHEMA, and SPECIFIC_TYPE_NAME are set to the <user-defined type name> of the user-defined type and CURRENT_TRANSFORM_GROUP is set to the CURRENT_TRANSFORM_GROUP_FOR_TYPE <user-defined type name>.

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

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

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

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

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

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

NOTE 31 – A *P* whose <SQL parameter mode> is IN can be a <value expression> that contains zero, one, or more <dynamic parameter specification>s. Thus:

— Every D_x maps to one and only one P_y.

— Several D_x instances can map to the same P_y.

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

d) The PARAMETER_MODE value in the descriptor for each D_x is set to the value from Table 11, "Codes associated with <parameter mode> in SQL/CLI", in ISO/IEC 9075-3, that indicates the <SQL parameter mode> of P_y.

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

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

Conformance Rules

The following restrictions apply for Core SQL:

None.

21.4 Implicit DESCRIBE OUTPUT USING clause

Function

Populate a specified descriptor area with information about the values returned by an execution of a specified SQL statement.

General Rules

1) Let *S* and *DESC* be a *SOURCE* and a *DESCRIPTOR* specified in the rules of this Subclause.

2) Let *HL* be the standard programming language of the invoking SQL-server.

3) The value of DYNAMIC_FUNCTION and DYNAMIC_FUNCTION_CODE in *DESC* are respectively a character string representation of the prepared statement and a numeric code that identifies the prepared statement.

4) A representation of the column descriptors of the <select list> columns for the prepared statement is stored in *DESC* as follows:

   a) Case:

      i) If there is a select source associated with *DESC*, then:

         1) Let *TBL* be the table defined by *S* and let *D* be the degree of *TBL*.

            Case:

            A) If the value of the statement attribute NEST DESCRIPTOR is *true* , then let NS_i, 1 (one) $\leq i \leq D$, be the number of subordinate descriptors of the descriptor for the *i*-th column of *T*.

            B) Otherwise, let NS_i, 1 (one) $\leq i \leq D$, be 0 (zero).

         2) TOP_LEVEL_COUNT is set to *D*. If *D* is 0 (zero), then let *TD* be 0 (zero); otherwise, let *TD* be $D + \sum_{i=1}^{D} (NS_i)$. COUNT is set to *TD*.

         3) Case:

            A) If some subset of *SL* is the primary key of *TBL*, then KEY_TYPE is set to 1 (one).

            B) If some subset of *SL* is the preferred key of *TBL*, then KEY_TYPE is set to 2.

            C) Otherwise, KEY_TYPE is set to 0 (zero).

      ii) Otherwise:

         1) Let *D* be 0 (zero). Let *TD* be 0 (zero).

         2) KEY_TYPE is set to 0 (zero).

   b) If *TD* is zero, then no item descriptor areas are set. Otherwise, the first *TD* item descriptor areas are set so that the *i*-th item descriptor area contains a descriptor of the *j*-th <dynamic parameter specification> such that:

      i) The descriptor for the first such column is assigned to the first descriptor area.

ii) The descriptor for the j+1-th column is assigned to the $i+NS_j$+1-th item descriptor area.

iii) If the value of the statement attribute NEST DESCRIPTOR is *true* , then the implicitly ordered subordinate descriptors for the j-th column are assigned to contiguous item descriptor areas starting at the i+1-th item descriptor area.

c) The descriptor of a column consists of values for LEVEL, TYPE, NULLABLE, NAME, UNNAMED, KEY_MEMBER, and other fields depending on the value of TYPE as described below. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values. The DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields are not relevant in this case.

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

ii) TYPE is set to a code as shown in Table 7, "Codes used for implementation data types in SQL/CLI", in ISO/IEC 9075-3, indicating the data type of the column or subordinate descriptor.

iii) Case:

1) If the value of LEVEL is 0 (zero), then:

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

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

C) Case:

I) If a <select list> column *C* is a member of a primary or preferred key of *TBL*, then KEY_MEMBER is set to 1 (one).

II) Otherwise, KEY_MEMBER is set to 0 (zero).

2) Otherwise:

A) NULLABLE is set to 1 (one).

B) Case:

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

Case:

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

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

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

C) KEY_MEMBER is set to 0 (zero).

iv) Case:

1) If TYPE indicates a <character string type>, then: LENGTH is set to the length or maximum length in characters of the character string and OCTET_LENGTH is set the maximum possible length in octets of the character string; if *HL* is C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string's collation.

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

3) If TYPE indicates a <binary string type>, then LENGTH is set to the maximum length in octets of the binary string.

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

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

6) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 9, "Codes associated with datetime data types in SQL/CLI", in ISO/IEC 9075-3, to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision> if either is applicable.

7) If TYPE indicates INTERVAL, then LENGTH is set to the length in positions of the interval type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 10, "Codes associated with <interval qualifier> in SQL/CLI", in ISO/IEC 9075-3, to indicate the specific <interval qualifier>, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and PRECISION is set to the <interval fractional seconds precision>, if applicable.

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

9) If TYPE indicates USER-DEFINED TYPE, then USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are set to the <user-defined type name> of the user-defined type. SPECIFIC_TYPE_CATALOG, SPECIFIC_TYPE_SCHEMA, and SPECIFIC_TYPE_NAME are set to the <user-defined type name> of the user-defined type and CURRENT_

TRANSFORM_GROUP is set to the CURRENT_TRANSFORM_GROUP_FOR_TYPE <user-defined type name>.

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

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

Conformance Rules

The following restrictions apply for Core SQL:

None.

21.5 Implicit EXECUTE USING and OPEN USING clauses

Function

Specify the rules for an implicit EXECUTE USING clause and an implicit OPEN USING clause.

General Rules

1) Let *T*, *S*, and *AS* be a *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT* specified in the rules of this Subclause.

2) Let *WPD*, *SRD*, and *SPD* be the wrapper parameter descriptor, server row descriptor, and server parameter descriptor, respectively, for *AS*.

3) *WPD* and *SPD* describe the <dynamic parameter specification>s and <dynamic parameter specification> values, respectively, for the statement being executed. Let *D* be the number of <dynamic parameter specification>s in *S*. Let *NSPD* be the value of COUNT for *SPD* and let *NWPD* be the value of COUNT for *WPD*.

   a) If *NSPD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.

   b) If *NWPD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.

   c) If *NWPD* is less than *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

   d) Let *NIDAL* be the number of item descriptor areas in *WPD* for which LEVEL is 0 (zero). If *NIDAL* is greater than *D*, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

   e) If the first *NWPD* item descriptor areas of *WPD* are not valid as specified in Subclause 21.1, "Description of MED item descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

   f) Let *AD* be the minimum of *NSPD* and *NWPD*.

   g) For each of the first *AD* item descriptor areas of *SPD*, if TYPE indicates DEFAULT, then:

      a) Let *TP*, *P*, and *SC* be the values of the TYPE, PRECISION, and SCALE fields, respectively, for the corresponding item descriptor area of *WPD*.

      b) The data type, precision, and scale of the described <dynamic parameter specification> value (or part thereof, if the item descriptor area is a subordinate descriptor) are set to *TP*, *P*, and *SC*, respectively, for the purposes of this invocation only.

   h) If the first *AD* item descriptor areas of *SPD* are not valid as specified in Subclause 21.1, "Description of MED item descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

i) For the first *AD* item descriptor areas in *SPD*:

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

  ii) If all of the following are true, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

    1) The value of the host variable addressed by INDICATOR POINTER is not negative.

    2) Either of the following is true:

      A) TYPE does not indicate ROW and the item descriptor area is not subordinate to an item descriptor area for which the value of the host variable addressed by the INDICATOR POINTER is not negative.

      B) TYPE indicates ARRAY or ARRAY LOCATOR.

      C) The value of the host variable addressed by DATA_POINTER is not a valid value of the data type represented by the item descriptor area.

j) If all of the following are true for any item descriptor area in the first *AD* item descriptor areas of *SPD*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

  i) DEFERRED is true for the item descriptor area.

  ii) Either of the following is true:

    1) The value of LEVEL is zero and TYPE indicates ROW or ARRAY.

    2) LEVEL is greater than 0 (zero).

    NOTE 32 – This rule states that a parameter whose type is ROW or ARRAY must be bound; it cannot be a deferred parameter.

k) For each item descriptor area whose LEVEL is 0 (zero) and for each of its subordinate descriptor areas, if any, for which DEFERRED is false in the first *AD* item descriptor areas of *SPD* and whose corresponding <dynamic parameter specification> has a <parameter mode> of PARAM MODE IN or PARAM MODE INOUT, refer to the corresponding <dynamic parameter specification> value as an immediate parameter value and refer to the corresponding <dynamic parameter specification> as an *immediate parameter*.

l) Let *IDA* be the *i*-th item descriptor area of *SPD* whose LEVEL value is 0 (zero). Let *SDT* be the data type represented by *IDA*. The associated value of *IDA* denoted by *SV*, is defined as follows.

  Case:

  i) If NULL is true for *IDA*, then *SV* is the null value.

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

  iii) Otherwise:

    1) Let *V* be the value of the host variable addressed by DATA_POINTER.

2) Case:

A) If TYPE indicates CHARACTER, then

Case:

I) If OCTET_LENGTH_POINTER is zero or if OCTET_LENGTH_POINTER is not zero and the value of the host variable addressed by OCTET_LENGTH_POINTER indicates NULL TERMINATED, then let *L* be the number of characters of *V* that precede the implementation-defined null character that terminates a C character string.

II) Otherwise, let *Q* be the value of the host variable addressed by OCTET_LENGTH_POINTER and let *L* be the number of characters wholly contained in the first *Q* octets of *V*.

B) Otherwise, let *L* be zero.

3) Let *SV* be *V* with effective data type *SDT*, as represented by the length value *L* and by the values of the TYPE, PRECISION, and SCALE fields.

m) Let *TDT* be the effective data type of the *i*-th immediate parameter as represented by the values of the TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields in the *i*-th item descriptor area of *WPD* for which the LEVEL value is 0 (zero), and all its subordinate descriptor areas.

n) Let *SDT* be the effective data type of the *i*-th bound parameter as represented by the values of the TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields in the corresponding item descriptor area of *SPD* for which the LEVEL is 0 (zero), and all its subordinate descriptor areas.

o) Case:

p) If *SDT* is a locator type, then let *TV* be the value *SV*.

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

1) Case:

A) If the <cast specification>

```
CAST ( SV AS TDT )
```

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

B) Otherwise:

I) If the <cast specification>

CAST (*SV* AS *TDT*)

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

II) If the <cast specification>

CAST (*SV* AS *TDT*)

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

III) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed and the result is the value *TV* of the *i*-th bound target.

2) Let *UDT* be the effective data type of the actual *i*-th immediate parameter, defined to be the data type represented by the values of the TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields that would automatically be set in the corresponding item descriptor area of *WPD* if POPULATE WPD was true.

3) Case:

A) If the <cast specification>

CAST (*TV* AS *UDT*)

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

B) Otherwise:

I) If the <cast specification>

CAST (*TV* AS *UDT*)

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

II) If the <cast specification>

CAST (*TV* AS *UDT*)

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

III) The <cast specification>

```
CAST ( TV AS UDT )
```

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

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

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

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

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

Case:

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

Case:

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

Case:

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

```
DT( )
```

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

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

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

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

s) If DEFERRED is true for at least one of the first *AD* item descriptor areas of *SPD*, then:

i) Let *PN* be the parameter number associated with the first such item descriptor area.

ii) *PN* becomes the deferred parameter number associated with *AS*.

iii) If *T* is 'EXECUTE', then *S* becomes the statement source associated with *AS*.

iv) An exception condition is raised: *foreign-data wrapper-specific condition — dynamic parameter value needed*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

21.6 Implicit CALL USING clause

Function

Specify the rules for an implicit CALL USING clause.

General Rules

1) Let *S* and *AS* be a *SOURCE* and an *ALLOCATED STATEMENT* specified in the rules of this Subclause.

2) Let *WPD* and *SPD* be the wrapper parameter descriptor and server row descriptor, respectively, for *AS*.

3) *WPD* and *SPD* describe the <dynamic parameter specification>s and <dynamic parameter specification> values, respectively, for the <call statement> being executed. Let *D* be the number of <dynamic parameter specification>s in *S*.

   a) Let *AD* be the value of the COUNT field of *SPD*. If *AD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor* count.

   b) For each item descriptor area in *SPD* whose LEVEL is 0 (zero) in the first *AD* item descriptor areas of *SPD*, and for all of their subordinate descriptor areas, refer to a <dynamic parameter specification> value whose corresponding item descriptor areas have a non-zero DATA_POINTER value and whose corresponding <dynamic parameter specification> has a <parameter mode> of PARAM MODE OUT or PARAM MODE INOUT as a bound target and refer to the corresponding <dynamic parameter specification> as a *bound parameter*.

   c) If any item descriptor area corresponding to a bound target in the first *AD* item descriptor areas of *SPD* is not valid as specified in Subclause 21.1, "Description of MED item descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

   d) Let *SDT* be the effective data type of the *i*-th bound parameter as represented by the values of the TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields in the *i*-th item descriptor area of *WPD* for which the LEVEL is 0 (zero) and all of its subordinate descriptor areas. Let *SV* be the value of the output parameter, with data type *SDT*.

   e) If TYPE indicates USER-DEFINED TYPE, then let the most specific type of the *i*-th bound parameter whose value is *SV* be represented by the values of the SPECIFIC_TYPE_CATALOG, SPECIFIC_TYPE_SCHEMA, and SPECIFIC_TYPE_NAME fields in the corresponding item descriptor area of *WPD*.

   f) Let *TYPE*, *OL*, *DP*, *IP*, and *LP* be the values of the TYPE, OCTET_LENGTH, DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields, respectively, in the item descriptor area of *SPD* corresponding to the *i*-th bound target (or part thereof, if the item descriptor area is a subordinate descriptor).

g) Case:

i) If TYPE indicates CHARACTER, then:

1) Let *UT* be the code value corresponding to CHARACTER VARYING as specified in Table 8, "Codes used for application data types in SQL/CLI", in ISO/IEC 9075-3.

2) Let *LV* be the implementation-defined maximum length for a CHARACTER VARYING data type.

ii) Otherwise, let *UT* be TYPE and let *LV* be 0 (zero).

h) Let *TDT* be the effective data type of the *i*-th bound target as represented by the type *UT*, the length value *LV*, and the values of the PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_ CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields in the corresponding item descriptor area of the SPD for which the LEVEL is 0 (zero) and all its subordinate descriptor areas.

i) Case:

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

1) If *SV* is not the null value, then a locator *L* that uniquely identifies *SV* is generated and the value *TV* of the *i*-th bound target is set to an implementation-dependent four-octet value that represents *L*.

2) Otherwise, the value *TV* of the *i*-th bound target is the null value.

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

Case:

1) If the <cast specification>

```
CAST ( SV AS TDT )
```

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

2) Otherwise:

A) If the <cast specification>

```
CAST ( SV AS TDT )
```

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

B) If the <cast specification>

```
CAST ( SV AS TDT )
```

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

C) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed and the result is the value *TV* of the *i*-th bound target.

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

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

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

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

Case:

A) If there is an applicable from-sql function, then let *FSF* be that from-sql function and let *FSFRT* be the <returns data type> of *FSF*.

Case:

I) If *FSFRT* is compatible with *TDT*, then the from-sql function *TSF* is effectively invoked with *SV* as its input parameter and the <return value> is the value *TV* of the *i*-th bound target.

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

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

j) Let *IDA* be the top-level item descriptor area corresponding to the *i*-th output parameter.

k) Case:

i) If TYPE indicates ROW, then

Case:

1) If *TV* is the null value, then

Case:

A) If *IP* is a null pointer for *IDA* or for any of the subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose type indicates ARRAY or ARRAY_LOCATOR, then an exception condition is raised: *data exception — null value, no indicator parameter*.

B) Otherwise, the value of the host variable addressed by *IP* for *IDA*, and that in all subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose TYPE indicates ARRAY or ARRAY_LOCATOR, is set to the appropriate 'Code' for SQL NULL DATA in Table 25, "Miscellaneous codes

used in SQL/CLI", in ISO/IEC 9075-3, and the values of variables addressed by *DP* and *LP* are implementation-dependent.

2) Otherwise, the *i*-th subordinate descriptor area of *IDA* is set to reflect the value of the *i*-th field of *TV* by applying General Rule , "3)k)" to the *i*-th subordinate descriptor area of *IDA* as *IDA*, the value of *i*-th field of *TV* as *TV*, the value of the *i*-th field of *SV* as *SV*, and the data type of the *i*-th field of *SV* as *SDT*.

ii) Otherwise,

Case:

1) If *TV* is the null value, then

Case:

A) If *IP* is a null pointer, then an exception condition is raised: *data exception — null value, no indicator parameter*.

B) Otherwise, the value of the host variable addressed by *IP* is set to the appropriate 'Code' for SQL NULL DATA in Table 25, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of the host variables addressed by *DP* and *LP* are implementation-dependent.

2) Otherwise:

A) If *IP* is not a null pointer, then the value of the host variable addressed by *IP* is set to 0 (zero).

B) Case:

I) If TYPE indicates CHARACTER or CHARACTER LARGE OBJECT, then:

1) If *TV* is a zero-length character string, then it is implementation-defined whether or not an exception condition is raised: *data exception — zero-length character string*.

2) The General Rules of Subclause 21.8, "Character string retrieval", are applied with *DP*, *TV*, *OL*, and *LP* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

II) If TYPE indicates BINARY LARGE OBJECT, then the General Rules of Subclause Subclause 21.9, "Binary large object string retrieval", are applied with *DP*, *TV*, *OL*, and *LP* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

III) If TYPE indicates ARRAY or ARRAY LOCATOR, and if RETURNED_ CARDINALITY_POINTER is not 0 (zero), then the value of the host variable addressed by RETURNED_CARDINALITY_POINTER is set to the cardinality of *TV*.

IV) Otherwise, the value of the host variable addressed by *DP* is set to *TV*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

21.7 Implicit FETCH USING clause

Function

Specify the rules for an implicit FETCH USING clause.

General Rules

1) Let *S* and *AS* be a *SOURCE* and an *ALLOCATED STATEMENT* specified in the rules of this Subclause.

2) Let *WRD* and *SRD* be the wrapper row descriptor and server row descriptor, respectively, associated with *AS*.

3) *WRD* and *SRD* describe the <select list> columns and <target specification>s, respectively, for the column values that are to be retrieved. Let *D* be the degree of the table defined by *S*.

   a) Let *AD* be the value of the COUNT field of *SRD*. If *AD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.

   b) For each item descriptor area in *SRD* whose LEVEL is 0 (zero) in the first AD item descriptor areas of *SRD*, and for all of their subordinate descriptor areas, refer to a <target specification> whose corresponding item descriptor areas have a non-zero DATA_POINTER as a *bound target* and refer to the corresponding <select list> column as a *bound column*.

   c) If any item descriptor area corresponding to a bound target in the first *AD* item descriptor areas of *SRD* is not valid as specified in Subclause 21.1, "Description of MED item descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

   d) Let *SDT* be the effective data type of the *i*-th bound column as represented by the values of the TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields in the *i*-th item descriptor area of *WRD* whose LEVEL is 0 (zero) and all of its subordinate descriptor areas.

   e) If TYPE indicates USER-DEFINED TYPE, then let the most specific type of the *i*-th bound column whose value is *SV* be represented by the values of the SPECIFIC_TYPE_CATALOG, SPECIFIC_TYPE_SCHEMA, and SPECIFIC_TYPE_NAME fields in the corresponding item descriptor area of *WRD*.

   f) Let *TYPE*, *OL*, *DP*, *IP*, and *LP* be the values of the TYPE, OCTET_LENGTH, DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields, respectively, in the item descriptor area of *SRD* corresponding to the *i*-th bound target (or part thereof, if the item descriptor area is a subordinate descriptor).

   g) Let *SV* be the value of the <select list> column, with data type *SDT*.

   h) Case:

      i) If TYPE indicates CHARACTER, then:

         1) Let *UT* be the code value corresponding to CHARACTER VARYING as specified in Table 7, "Codes used for implementation data types in SQL/CLI", in ISO/IEC 9075-3.

2) Let *LV* be the implementation-defined maximum length for a CHARACTER VARYING data type.

ii) Otherwise, let *UT* be TYPE and let *LV* be 0 (zero).

i) Let *TDT* be the effective data type of the *i*-th bound target as represented by the type UT, the length value *LV*, and the values of the PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields in the item descriptor area of *SRD* whose LEVEL is 0 (zero) and all of its subordinate descriptor areas.

j) Let *LTDT* be the data type on the last fetch of the *i*-th bound target, if any. If any of the following is true, then is implementation-defined whether or not an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

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

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

iii) If *LTDT* and *TDT* both identify an array type and only one of LTDT and TDT is an array locator.

iv) If *LTDT* and *TDT* both identify a user-defined type and only one of LTDT and TDT is a user-defined type locator.

k) Case:

i) If TDT is a locator type, then:

1) If *SV* is not the null value, then a locator *L* that uniquely identifies *SV* is generated and the value *TV* of the *i*-th bound target is set to an implementation-dependent four-octet value that represents *L*.

2) Otherwise, the value *TV* of the *i*-th bound target is the null value.

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

Case:

1) If the <cast specification>

```
CAST ( SV AS TDT )
```

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

2) Otherwise:

A) If the <cast specification>

```
CAST ( SV AS TDT )
```

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

B) If the <cast specification>

CAST (*SV* AS *TDT*)

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

C) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed and the result is the value *TV* of the *i*-th bound target.

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

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

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

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

Case:

A) If there is an applicable from-sql function, then let *FSF* be that from-sql function and let *FSFRT* be the <returns data type> of *FSF*.

Case:

I) If *FSFRT* is compatible with *TDT*, then the from-sql function *TSF* is effectively invoked with *SV* as its input parameter and the <return value> is the value *TV* of the *i*-th bound target.

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

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

l) Let *IDA* be the top-level item descriptor area corresponding to the *i*-th bound column.

m) Case:

i) If TYPE indicates ROW, then

Case:

1) If *TV* is the null value, then

Case:

A) If *IPE* is a null pointer for *IDA* or for any of the subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose type indicates ARRAY or ARRAY_LOCATOR, then an exception condition is raised: *data exception — null value, no indicator parameter*.

B) Otherwise, the value of the host variable addressed by *IPE* for *IDA*, and that in all subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose TYPE indicates ARRAY or ARRAY_LOCATOR, is set to the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of variables addressed by *DPE* and *LPE* are implementation-dependent.

2) Otherwise, the *i*-th subordinate descriptor area of *IDA* is set to reflect the value of the i-th field of *TV* by applying General Rule , "3)m)" to the *i*-th subordinate descriptor area of *IDA* as *IDA*, the value of *i*-th field of *TV* as *TV*, the value of the *i*-th field of *SV* as *SV*, and the data type of the *i*-th field of *SV* as *SDT*.

ii) Otherwise,

Case:

1) If *TV* is the null value, then

Case:

A) If *IPE* is a null pointer, then an exception condition is raised: *data exception — null value, no indicator parameter*.

B) Otherwise, the value of the host variable addressed by *IPE* is set to the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of the host variables addressed by *DPE* and *LPE* are implementation-dependent.

2) Otherwise:

A) If *IPE* is not a null pointer, then the value of the host variable addressed by *IPE* is set to 0 (zero).

B) Case:

I) If TYPE indicates CHARACTER or CHARACTER LARGE OBJECT, then:

1) If *TV* is a zero-length character string, then it is implementation-defined whether or not an exception condition is raised: *data exception — zero-length character string*.

2) The General Rules of Subclause 21.8, "Character string retrieval", are applied with *DPE*, *TV*, *OL*, and *LPE* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

II) If TYPE indicates BINARY LARGE OBJECT, then the General Rules of Subclause 21.9, "Binary large object string retrieval", are applied with *DPE*, *TV*, *OL*, and *LPE* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

III) If TYPE indicates ARRAY or ARRAY LOCATOR, and if RETURNED_CARDINALITY_POINTER is not a null pointer, then the value of the host variable addressed by RETURNED_CARDINALITY_POINTER is set to the cardinality of *TV*.

IV) Otherwise, the value of the host variable addressed by *DPE* is set to *TV* and the value of the host variable addressed by *LPE* is implementation-dependent.

Conformance Rules

The following restrictions apply for Core SQL:

None.

21.8 Character string retrieval

Function

Specify the rules for retrieving character string values.

General Rules

1) Let *T*, *V*, *TL*, and *RL* be a *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH* specified in an application of this Subclause.

2) If *TL* is not greater than zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

3) Let *L* be the length in octets of *V*.

4) If *RL* is not a null pointer, then *RL* is set to *L*.

5) Case:

   a) If null termination is false for the SQL-server, then:

      i) If *L* is not greater than *TL*, then the first *L* octets of *T* are set to *V* and the values of the remaining octets of *T* are implementation-dependent.

      ii) Otherwise, *T* is set to the first *TL* octets of *V* and a completion condition is raised: *warning — string data, right truncation*.

   b) Otherwise, let *NB* be the length in octets of a null terminator in the character set of *T*.

      Case:

      i) If *L* is not greater than (*TL*−*NB*) then the first (*L*+*NB*) octets of *T* are set to *V*, concatenated with a single implementation-defined null character that terminates a C character string. The values of the remaining characters of *T* are implementation-dependent.

      ii) Otherwise, *T* is set to the first (*TL*−*NB*) octets of *V* concatenated with a single implementation-defined null character that terminates a C character string and a completion condition is raised: *warning — string data, right truncation*.

21.9 Binary large object string retrieval

Function

Specify the rules for retrieving BINARY LARGE OBJECT string values.

General Rules

1) Let *T*, *V*, *TL*, and *RL* be a *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH* specified in an application of this Subclause.

2) If *TL* is not greater than zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

3) Let *L* be the length in octets of *V*.

4) If *RL* is not a null pointer, then *RL* is set to *L*.

5) Case:

   a) If *L* is not greater than *TL*, then the first *L* octets of *T* are set to *V* and the values of the remaining octets of *T* are implementation-dependent.

   b) Otherwise, *T* is set to the first *TL* octets of *V* and a completion condition is raised: *warning — string data, right truncation*.

21.10 Deferred parameter check

Function

Check for the existence of deferred dynamic parameters when accessing an SQL/MED descriptor.

General Rules

1) Let *DA* be a *DESCRIPTOR AREA* specified in an application of this Subclause.

2) Let *L1* be the set of all allocated SQL-statements associated with *DA*.

3) Let *L2* be the set of all allocated SQL-statements in *L1* that have an associated deferred parameter number.

4) Let *L3* be the set of all SQL/MED descriptor areas that are either the server parameter descriptor for, or the wrapper parameter descriptor associated with, an allocated SQL-statement in *L2*.

5) If *DA* is contained in *L3*, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error*.

21.11 MED-specific status codes

Some of the conditions that can occur during the execution of Foreign-data wrapper interface routines are MED-specific. The corresponding status codes are listed in Table 29, "SQLSTATE class and subclass values for SQL/MED-specific conditions".

Table 29—SQLSTATE class and subclass values for SQL/MED-specific conditions

Category	Condition	Class	Subcondition	Subclass
X	datalink exception	HW	(*no subclass*)	000
			external file not linked	001
			external file already linked	002
X	foreign-data wrapper-specific condition	HV	(*no subclass*)	000
			column name not found	005
			dynamic parameter value needed	002
			execution handle does not match reply handle	003
			function sequence error	010
			inconsistent descriptor information	021
			invalid attribute value	024
			invalid column name	007
			invalid column number	008
			invalid data type	004
			invalid descriptor field identifier	091
			invalid handle	00B
			invalid option index	00C
			invalid option name	00D
			invalid string length or buffer length	090
			invalid use of null pointer	009
			limit on number of handles exceeded	0014
			memory allocation error	001
			option name not found	00J
			reply handle	00K
			unable to create execution	00L
			unable to create reply	00M
			unable to establish connection to foreign server	00N

21.12 Tables used with SQL/MED

The tables contained in this Subclause are used to specify the codes used by the various SQL/MED routines.

Table 30—Codes used for <table reference> types

<table reference> type	Code
TABLE_NAME	1

Table 31—Codes used for <value expression> types

<value expression> type	Code
COLUMN_NAME	1

Table 32—Codes used for SQL/MED diagnostic fields

Field	Code	Type
CLASS_ORIGIN	1	Status
MESSAGE_LENGTH	2	Status
MESSAGE_OCTET_LENGTH	3	Status
MESSAGE_TEXT	4	Status
MORE	5	Header
NATIVE_CODE	6	Status
NUMBER	7	Header
RETURNCODE	8	Header
SQLSTATE	9	Status
SUBCLASS_ORIGIN	10	Status
Implementation-defined diagnostics header field	< 0	Header
Implementation-defined diagnostics status field	< 0	Status

Table 33—Codes used for SQL/MED descriptor fields

Field	Code	SQL Item Descriptor Name	Type
CARDINALITY	1040	CARDINALITY	Item
CHARACTER_SET_CATALOG	1018	CHARACTER_SET_CATALOG	Item
CHARACTER_SET_NAME	1020	CHARACTER_SET_NAME	Item

Table 33—Codes used for SQL/MED descriptor fields (Cont.)

Field	Code	SQL Item Descriptor Name	Type
CHARACTER_SET_SCHEMA	1019	CHARACTER_SET_SCHEMA	Item
COLLATION_CATALOG	1015	COLLATION_CATALOG	Item
COLLATION_NAME	1017	COLLATION_NAME	Item
COLLATION_SCHEMA	1016	COLLATION_SCHEMA	Item
COUNT	1001	COUNT	Header
CURRENT_TRANSFORM_GROUP	1039	(Not applicable)	Item
DATA	1050	DATA	Item
DATA_POINTER	1010	DATA	Item
DATETIME_INTERVAL_CODE	1007	DATETIME_INTERVAL_CODE	Item
DATETIME_INTERVAL_PRECISION	26	DATETIME_INTERVAL_PRECISION	Item
DEGREE	1041	DEGREE	Item
DYNAMIC_FUNCTION	1031	DYNAMIC_FUNCTION	Header
DYNAMIC_FUNCTION_CODE	1032	DYNAMIC_FUNCTION_CODE	Header
INDICATOR	1051	INDICATOR	Item
INDICATOR_POINTER	1009	INDICATOR	Item
KEY_MEMBER	1030	KEY_MEMBER	Item
KEY_TYPE	1029	KEY_TYPE	Header
LENGTH	1003	LENGTH	Item
LEVEL	1042	LEVEL	Item
NAME	1011	NAME	Item
NULLABLE	1008	NULLABLE	Item
OCTET_LENGTH	1013	OCTET_LENGTH	Item
OCTET_LENGTH_POINTER	1004	Both OCTET_LENGTH (input) and RETURNED_OCTET_LENGTH (output)	Item
PARAMETER_MODE	1021	PARAMETER_MODE	Item
PARAMETER_ORDINAL_POSITION	1022	PARAMETER_ORDINAL_POSITION	Item
PARAMETER_SPECIFIC_CATALOG	1023	PARAMETER_SPECIFIC_CATALOG	Item
PARAMETER_SPECIFIC_NAME	1025	PARAMETER_SPECIFIC_NAME	Item
PARAMETER_SPECIFIC_SCHEMA	1024	PARAMETER_SPECIFIC_SCHEMA	Item
PRECISION	1005	PRECISION	Item
RETURNED_CARDINALITY	1052	RETURNED_CARDINALITY	Item
RETURNED_CARDINALITY_POINTER	1043	RETURNED_CARDINALITY_POINTER	Item

Table 33—Codes used for SQL/MED descriptor fields (Cont.)

Field	Code	SQL Item Descriptor Name	Type
RETURNED_OCTET_LENGTH	1053	Both OCTET_LENGTH (input) and RETURNED_OCTET_LENGTH (output)	Item
SCALE	1006	SCALE	Item
SCOPE_CATALOG	1033	SCOPE_CATALOG	Item
SCOPE_NAME	1034	SCOPE_NAME	Item
SCOPE_SCHEMA	1035	SCOPE_SCHEMA	Item
SPECIFIC_TYPE_CATALOG	1036	(Not applicable)	Item
SPECIFIC_TYPE_NAME	1038	(Not applicable)	Item
SPECIFIC_TYPE_SCHEMA	1037	(Not applicable)	Item
TOP_LEVEL_COUNT	1044	TOP_LEVEL_COUNT	Header
TYPE	1002	TYPE	Item
UNNAMED	1012	UNNAMED	Item
USER_DEFINED_TYPE_CATALOG	1026	USER_DEFINED_TYPE_CATALOG	Item
USER_DEFINED_TYPE_NAME	1028	USER_DEFINED_TYPE_NAME	Item
USER_DEFINED_TYPE_SCHEMA	1027	USER_DEFINED_TYPE_SCHEMA	Item
Implementation-defined SQL/MED descriptor header field	0 (zero) through 999, or ≥ 1200, excluding values defined in this table	Implementation-defined SQL/MED descriptor header field	Header
Implementation-defined SQL/MED descriptor item field	0 (zero) through 999, or ≥ 1200, excluding values defined in this table	Implementation-defined SQL/MED descriptor item field	Item

Table 34—Codes used for SQL/MED handle types

Handle type	Code
ExecutionHandle	1
FSConnectionHandle	2
ReplyHandle	3
RequestHandle	4
DataHandle	5
ServerHandle	6
TableReferenceHandle	7
UserHandle	8
ValueExpressionHandle	9
WrapperHandle	10
WrapperEnvHandle	11
DescriptorHandle	12

Table 35—Ability to retrieve SQL/MED descriptor fields

	May be retrieved			
Field	SRD	WRD or TRD	SPD	WPD
CARDINALITY	No		No	
CHARACTER_SET_CATALOG				
CHARACTER_SET_NAME				
CHARACTER_SET_SCHEMA				
COLLATION_CATALOG				
COLLATION_NAME				
COLLATION_SCHEMA				
COUNT				
CURRENT_TRANSFORM_GROUP				
DATA		No		No
DATA_POINTER		No		No
DATETIME_INTERVAL_CODE				
DATETIME_INTERVAL_PRECISION				
DEGREE	No		No	
DYNAMIC_FUNCTION	No		No	
DYNAMIC_FUNCTION_CODE	No		No	

Table 35—Ability to retrieve SQL/MED descriptor fields (Cont.)

Field	May be retrieved			
	SRD	WRD or TRD	SPD	WPD
INDICATOR		No		No
INDICATOR_POINTER		No		No
KEY_MEMBER	No		No	No
KEY_TYPE	No		No	No
LENGTH				
LEVEL				
NAME				
NULLABLE				
OCTET_LENGTH				
OCTET_LENGTH_POINTER		No		No
PARAMETER_MODE	No		No	
PARAMETER_ORDINAL_POSITION	No		No	
PARAMETER_SPECIFIC_CATALOG	No		No	
PARAMETER_SPECIFIC_NAME	No		No	
PARAMETER_SPECIFIC_SCHEMA	No		No	
PRECISION				
RETURNED_CARDINALITY		No		No
RETURNED_CARDINALITY_POINTER		No		No
RETURNED_OCTET_LENGTH		No		No
SCALE				
SCOPE_CATALOG				
SCOPE_NAME				
SCOPE_SCHEMA				
SPECIFIC_TYPE_CATALOG				
SPECIFIC_TYPE_NAME				
SPECIFIC_TYPE_SCHEMA				
TOP_LEVEL_COUNT				
TYPE				
UNNAMED				
USER_DEFINED_TYPE_CATALOG				
USER_DEFINED_TYPE_NAME				

Table 35—Ability to retrieve SQL/MED descriptor fields (Cont.)

Field	May be retrieved			
	SRD	WRD or TRD	SPD	WPD
USER_DEFINED_TYPE_SCHEMA				
Implementation-defined SQL/MED descriptor header field	ID	ID	ID	ID
Implementation-defined SQL/MED descriptor item field	ID	ID	ID	ID

Table 36—Ability to set SQL/MED descriptor fields

Field	May be set			
	SRD	WRD or TRD	SPD	WPD
CARDINALITY	No	No	No	
CHARACTER_SET_CATALOG		No		
CHARACTER_SET_NAME		No		
CHARACTER_SET_SCHEMA		No		
COLLATION_CATALOG		No		
COLLATION_NAME		No		
COLLATION_SCHEMA		No		
COUNT		No		
CURRENT_TRANSFORM_GROUP	No	No	No	No
DATA		No		
DATA_POINTER		No		
DATETIME_INTERVAL_CODE		No		
DATETIME_INTERVAL_PRECISION		No		
DEGREE	No	No	No	
DYNAMIC_FUNCTION	No	No	No	No
DYNAMIC_FUNCTION_CODE	No	No	No	No
INDICATOR		No		No
INDICATOR_POINTER		No		No
KEY_MEMBER	No	No	No	No
KEY_TYPE	No	No	No	No
LENGTH		No		

Table 36—Ability to set SQL/MED descriptor fields (Cont.)

Field	May be set			
	SRD	WRD or TRD	SPD	WPD
LEVEL		No		
NAME		No		
NULLABLE		No		
OCTET_LENGTH		No		
OCTET_LENGTH_POINTER		No		No
PARAMETER_MODE	No	No	No	
PARAMETER_ORDINAL_POSITION	No	No	No	
PARAMETER_SPECIFIC_CATALOG	No	No	No	
PARAMETER_SPECIFIC_NAME	No	No	No	
PARAMETER_SPECIFIC_SCHEMA	No	No	No	
PRECISION		No		
RETURNED_CARDINALITY		No		No
RETURNED_CARDINALITY_POINTER		No		No
RETURNED_OCTET_LENGTH		No		No
SCALE		No		
SCOPE_CATALOG		No		
SCOPE_NAME		No		
SCOPE_SCHEMA		No		
SPECIFIC_TYPE_CATALOG	No	No	No	No
SPECIFIC_TYPE_NAME	No	No	No	No
SPECIFIC_TYPE_SCHEMA	No	No	No	No
TOP_LEVEL_COUNT		No		
TYPE		No		
UNNAMED		No		
USER_DEFINED_TYPE_CATALOG		No		
USER_DEFINED_TYPE_NAME		No		
USER_DEFINED_TYPE_SCHEMA		No		
Implementation-defined SQL/MED descriptor header field	ID	ID	ID	ID
Implementation-defined SQL/MED descriptor item field	ID	ID	ID	ID

Table 37—SQL/MED descriptor field default values

Field	Default values			
	SRD	WRD or TRD	APD	WPD
CARDINALITY				
CHARACTER_SET_CATALOG				
CHARACTER_SET_NAME				
CHARACTER_SET_SCHEMA				
COLLATION_CATALOG				
COLLATION_NAME				
COLLATION_SCHEMA				
COUNT	0 (zero)		0 (zero)	
CURRENT_TRANSFORM_GROUP				
DATA				
DATA_POINTER	Null		Null	
DATETIME_INTERVAL_CODE				
DATETIME_INTERVAL_PRECISION				
DEGREE				
DYNAMIC_FUNCTION				
DYNAMIC_FUNCTION_CODE				
INDICATOR				
INDICATOR_POINTER	Null		Null	
KEY_MEMBER				
KEY_TYPE				
LENGTH				
LEVEL	0 (zero)			
NAME				
NULLABLE				
OCTET_LENGTH				
OCTET_LENGTH_POINTER	Null		Null	
PARAMETER_MODE				
PARAMETER_ORDINAL_POSITION				
PARAMETER_SPECIFIC_CATALOG				

Table 37—SQL/MED descriptor field default values (Cont.)

Field	Default values			
	SRD	WRD or TRD	APD	WPD
PARAMETER_SPECIFIC_NAME				
PARAMETER_SPECIFIC_ SCHEMA				
PRECISION				
RETURNED_CARDINALITY				
RETURNED_CARDINALITY_ POINTER	Null		Null	
RETURNED_OCTET_LENGTH				
SCALE				
SCOPE_CATALOG				
SCOPE_NAME				
SCOPE_SCHEMA				
SPECIFIC_TYPE_CATALOG				
SPECIFIC_TYPE_NAME				
SPECIFIC_TYPE_SCHEMA				
TOP_LEVEL_COUNT	0 (zero)		0 (zero)	
TYPE				
UNNAMED				
USER_DEFINED_TYPE_ CATALOG				
USER_DEFINED_TYPE_NAME				
USER_DEFINED_TYPE_ SCHEMA				
Implementation-defined SQL/MED descriptor header field	ID	ID	ID	ID
Implementation-defined SQL/MED descriptor item field	ID	ID	ID	ID

22 Foreign-data wrapper interface routines

22.1 <foreign-data wrapper routine>

Function

Describe a generic foreign-data wrapper routine.

Format

```
<foreign-data wrapper routine> ::=
      <foreign-data wrapper routine name>
      <foreign-data wrapper parameter list>
      [ <foreign-data wrapper returns clause> ]

<foreign-data wrapper routine name> ::=
        AllocDescriptor
      | AllocWrapperEnv
      | Close
      | ConnectForeignServer
      | FreeDescriptor
      | FreeExecutionHandle
      | FreeForeignRequest
      | FreeFSConnection
      | FreeReplyHandle
      | FreeWrapperEnv
      | GetAuthorizationID
      | GetDescriptor
      | GetDiagnostics
      | GetForeignTable
      | GetForeignTableColumn
      | GetForeignTableColumnOption
      | GetForeignTableColumnOptionValueByName
      | GetForeignTableColumnType
      | GetForeignTableOption
      | GetForeignTableOptionValueByName
      | GetForeignTableServerName
      | GetNumberOfForeignTableColumns
      | GetNumberOfForeignTableColumnOptions
      | GetNumberOfForeignTableOptions
      | GetNumberOfReplySelectListElements
      | GetNumberOfReplyTableReferences
      | GetNumberOfSelectListElements
      | GetNumberOfServerOptions
      | GetNumberOfTableReferenceElements
      | GetNumberOfUserOptions
      | GetNumberOfWrapperOptions
      | GetOptions
      | GetReplySelectListElement
      | GetReplyTableReference
      | GetSelectListElement
      | GetSelectListElementType
      | GetServerName
      | GetServerOption
      | GetServerOptionValueByName
      | GetServerType
```

```
        | GetServerVersion
        | GetSPDHandle
        | GetSRDHandle
        | GetTableReferenceElement
        | GetTableReferenceElementType
        | GetTableReferenceTableName
        | GetTRDHandle
        | GetUserOption
        | GetUserOptionValueByName
        | GetValueExpressionColumnName
        | GetWPDHandle
        | GetSRDHandle
        | GetWrapperLibraryName
        | GetWrapperName
        | GetWrapperOption
        | GetWrapperOptionValueByName
        | InitForeignRequest
        | Iterate
        | Open
        | ReOpen
        | RetrieveStatistics
        | SetDescriptor
        | TransmitForeignRequest

<foreign-data wrapper parameter list> ::=
       <left paren> <foreign-data wrapper parameter declaration>
       [ { <comma> <foreign-data wrapper parameter declaration> }... ] <right paren>

<foreign-data wrapper parameter declaration> ::=
       <foreign-data wrapper parameter name>
       <foreign-data wrapper parameter mode>
       <foreign-data wrapper parameter data type>

<foreign-data wrapper parameter name> ::= !! See the individual foreign-data wrapper
routine definitions

<foreign-data wrapper parameter mode> ::=
         IN
       | OUT
       | INOUT

<foreign-data wrapper parameter data type> ::=
         INTEGER
       | SMALLINT
       | ANY
       | CHARACTER <left paren> <length> <right paren>

<foreign-data wrapper returns clause> ::= RETURNS SMALLINT
```

Syntax Rules

1) <foreign-data wrapper routine> is a predefined routine written in a standard programming language that is invoked by a compilation unit of the same standard programming language. Let *HL* be that standard programming language. *HL* shall be one of Ada, C, COBOL, Fortran, MUMPS, Pascal, or PL/I.

2) <foreign-data wrapper routine> that contains a <foreign-data wrapper returns clause> is called a *foreign-data wrapper function*. A <foreign-data wrapper routine> that does not contain a <foreign-data wrapper returns clause> is called a *foreign-data wrapper procedure*.

3) For each foreign-data wrapper function *WF*, there is a corresponding foreign-data wrapper procedure *WP*, with the same <foreign-data wrapper routine name>. The <foreign-data wrapper parameter list> for *WP* is the same as the <foreign-data wrapper parameter list> for *WF* but with the following additional <foreign-data wrapper parameter declaration>:

```
ReturnCode OUT SMALLINT
```

4) *HL* shall support either the invocation of *WF* or the invocation of *WP*. It is implementation-defined which is supported.

5) Case:

   a) If <foreign-data wrapper parameter mode> is IN, then the parameter is an *input parameter*. The value of an input argument is established when a foreign-data wrapper routine is invoked.

   b) If <foreign-data wrapper parameter mode> is OUT, then the parameter is an *output parameter*. The value of an output argument is established when a foreign-data wrapper routine is executed.

   c) If <foreign-data wrapper parameter mode> is INOUT, then the parameter is both an input parameter and an output parameter.

6) Let *WR* be a <foreign-data wrapper routine> and let *RN* be its <foreign-data wrapper routine name>. Let *RNU* be the value of `UPPER(`*RN*`)`.

   Case:

   a) If *HL* supports case sensitive routine names, then the name used for the invocation of *WR* shall be *RN*.

   b) If *HL* does not support <simple Latin lower case letter>s, then the name used for the invocation of *WR* shall be *RNU*.

   c) If *HL* does not support case sensitive routine names, then the name used for the invocation of *WR* shall be *RN* or *RNU*.

7) Let operative data type correspondence table be the data type correspondence table for *HL* as specified in Subclause 15.4, "Data type correspondences". Refer to the two columns of the operative data type correspondence table as the "SQL data type column" and the "host data type column".

8) Let *TI*, *TS*, *TC*, and *TV* be the types listed in the host data type column for the rows that contains INTEGER, SMALLINT, CHARACTER(L) and CHARACTER VARYING(L), respectively, in the SQL data type column.

   a) If *TS* is "None", then let *TS* = *TI*.

   b) If *TC* is "None", then let *TC* = *TV*.

   c) For each parameter *P*,

      Case:

      i) If the foreign-data wrapper parameter data type is INTEGER, then the type of the corresponding argument shall be *TI*.

ii) If the foreign-data wrapper parameter data type is SMALLINT, then the type of the corresponding argument shall be *TS*.

iii) If the foreign-data wrapper parameter data type is CHARACTER(*L*), then the type of the corresponding argument shall be *TC*.

iv) If the foreign-data wrapper parameter data type is ANY, then

Case:

1) If *HL* is C, then the type of the corresponding argument shall be "**void**".

2) Otherwise, the type of the corresponding argument shall be any type (other than 'None') listed in the host data type column.

d) If the foreign-data wrapper routine is a foreign-data wrapper function, then the type of the returned value is *TS*.

Access Rules

None.

General Rules

1) The rules for invocation of the <foreign-data wrapper routine> are specified in Subclause 22.2, "<foreign-data wrapper routine> invocation".

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not specify <foreign-data wrapper routine>.

22.2 <foreign-data wrapper routine> invocation

Function

Specify the rules for invocation of a <foreign-data wrapper routine>.

Syntax Rules

1) Let *HL* be the standard programming language of *CP*, the caller of a <foreign-data wrapper routine>.

2) A foreign-data wrapper function or foreign-data wrapper procedure is invoked by the *HL* mechanism for invoking functions or procedures, respectively.

3) Let *RN* be the <foreign-data wrapper routine name> of the <foreign-data wrapper routine> invoked by *CP*. The number of arguments provided in the invocation shall be the same as the number of <foreign-data wrapper parameter declaration>s for *RN*.

4) Let *DA* be the data type of the *i*-th argument in the invocation and let *DP* be the <foreign-data wrapper parameter data type> of the *i*-th <foreign-data wrapper parameter declaration> of *RN*. *DA* shall be the *HL* equivalent of *DP* as specified by the rules of Subclause 22.1, "<foreign-data wrapper routine>".

Access Rules

None.

General Rules

1) If the value of any input argument provided by *CP* falls outside the set of allowed values of the data type of the parameter, or if the value of any output argument resulting from the execution of the <foreign-data wrapper routine> falls outside the set of values supported by *CP* for that parameter, then the effect is implementation-defined.

2) When the <foreign-data wrapper routine> is called by *CP*:

   a) The values of all input arguments to *RN* are established.

   b) *RN* is invoked.

3) Case:

   a) If the <foreign-data wrapper routine> is a foreign-data wrapper function, then:

      i) The values of all output arguments are established.

      ii) Let *RC* be the return value.

   b) If the <foreign-data wrapper routine> is a foreign-data wrapper procedure, then:

      i) The values of all output arguments are established except for the argument associated with the ReturnCode parameter.

      ii) Let *RC* be the argument associated with the ReturnCode parameter.

4) Case:

a) If *RN* executed successfully, then:

i) Either a completion condition is raised: *successful completion*, or a completion condition is raised: *no data*.

ii) Case:

1) If a completion condition is raised: *successful completion*, then *RC* is set to indicate **Success**.

2) If a completion condition is raised: *no data*, then *RC* is set to indicate **No data found**.

b) If *RN* did not execute successfully, then:

i) All changes made to SQL-data or schemas by the execution of *RN* are canceled.

ii) One or more exception conditions are raised as determined by the General Rules of this and other Subclauses of this part of ISO/IEC 9075 or by implementation-defined rules.

iii) Case:

1) If an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*, then *RC* is set to indicate **Invalid handle**.

2) Otherwise, *RC* is set to indicate **Error**.

Conformance Rules

None.

22.3 AllocDescriptor

Function

Allocate a descriptor area and assign a handle to it.

Definition

```
AllocDescriptor (
      FSConnectionHandle    IN       INTEGER,
      MaxDetailAreas        IN       SMALLINT,
      DescriptorHandle      OUT      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle and let *MDA* be the value of MaxDetailAreas.

2) If *FSCH* does not identify an allocated FS-connection, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *C* be the allocated FS-connection identified by *FSCH*.

4) The diagnostics area associated with *C* is emptied.

5) Case:

   a) If there is no established FS-connection associated with *C*, then DescriptorHandle is set to zero and an exception condition is raised: *connection exception — connection does not exist.*

   b) Otherwise, let *EC* be the established FS-connection associated with *C*.

6) If the maximum number of SQL/MED descriptor areas that can be allocated at one time has already been reached, then DescriptorHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — limit on number of handles exceeded.*

7) Case:

   a) If the memory requirements to manage an SQL/MED descriptor area having *MDA* item descriptor areas cannot be satisfied, then OutputHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error.*

   b) If the resources to manage an SQL/MED descriptor area cannot be allocated for implementation-defined reasons, then OutputHandle is set to zero and an implementation-defined exception condition is raised.

   c) Otherwise, the resources to manage an SQL/MED descriptor area are allocated and are referred to as an allocated SQL/MED descriptor area. The allocated SQL/MED descriptor area is associated with *C* and is assigned a unique value that is returned in OutputHandle. All fields of the allocated SQL/MED descriptor area are set to the default values for an SQL/MED descriptor area as specified in Table 37, "SQL/MED descriptor field default values". Fields in the SQL/MED item descriptor areas not set to a default value are initially undefined.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.4 AllocWrapperEnv

Function

Allocate a foreign-data wrapper environment and assign a handle to it.

Definition

```
AllocWrapperEnv (
      WrapperHandle         IN       INTEGER,
      WrapperEnvHandle      OUT      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *WH* be the value of WrapperHandle.

2) If *WH* does not identify an allocated foreign-wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) If the maximum number of foreign-data wrapper environments that can be allocated at one time has already been reached, then an exception condition is raised: *foreign-data wrapper-specific condition — limit on number of handles exceeded.* A skeleton FDW-environment is allocated and is assigned a unique value that is returned in WrapperEnvHandle.

4) Case:

   a) If the memory requirements to manage an foreign-data wrapper environment cannot be satisfied, then WrapperEnvHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error.*

      NOTE 33 – No diagnostic information is generated in this case, as there is no valid WrapperEnvHandle that can be used to obtain diagnostics information.

   b) If the resources to manage an foreign-data wrapper environment cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised. A skeleton FDW-environment is allocated and is assigned a unique value that is returned in WrapperEnvHandle.

   c) Otherwise, the resources to manage an foreign-data wrapper environment are allocated and are referred to as an *allocated FDW-environment.* The allocated FDW-environment is assigned a unique value that is returned in WrapperEnvHandle.

5) If an exception condition is raised in any of the routines invoked in any of the gollowing General Rules, then the diagnostics records returned by the invoked routines are transferred to the diagnostics area associated with the WrapperEnvHandle and further processing of this routine is terminated as if the exception condition had been raised in this routine.

6) Let *WN* be the WrapperName that would be returned by an invocation of the GetWrapperName routine with *WH* as the WrapperHandle parameter.

7) Let *WL* be the WrapperLibraryName that would be returned by an invocation of the GetWrapperLibraryName routine with *WH* as the WrapperHandle parameter.

Conformance Rules

None.

22.5 Close

Function

Close an FDW-execution.

Definition

```
Close (
      ExecutionHandle       IN       INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an opened FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error*.

3) Let *E* be the opened FDW-execution identified by *EH*.

4) Case:

   a) If there is no open cursor associated with *E*, then an exception condition is raised: *invalid cursor state*.

   b) Otherwise:

      i) The open cursor associated with *E* is placed in the closed state and its copy of the select source is destroyed.

      ii) Any fetched row associated with *E* is removed from association with *E*.

5) *EH* is reset to be an allocated FDW-execution.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.6 ConnectForeignServer

Function

Establish a connection to a foreign server and assign a handle to it.

Definition

```
ConnectForeignServer (
      WrapperEnvHandle      IN      INTEGER,
      ServerHandle          IN      INTEGER,
      UserHandle            IN      INTEGER,
      FSConnectionHandle    OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) If WrapperEnvHandle does not identify an allocated FDW-environment, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

2) Let *SH* be the value of ServerHandle.

3) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

4) Let *UH* be the value of UserHandle.

5) If *UH* does not identify an allocated user mapping description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

6) If an exception condition is raised in any of the routines invoked in any of the following General Rules, then the diagnostics records returned by the invoked routines are transferred to the diagnostics area associated with the WrapperEnvHandle and further processing of this routine is terminated as if the exception condition had been raised in this routine.

7) Let *UN* be the AuthorizationId that would be returned by an invocation of the GetAuthorizationId routine with *UH* as the UserHandle parameter.

8) Let *SN* be the ServerName that would be returned by an invocation of the GetServerName routine with *SH* as the ServerHandle parameter.

9) Let *ST* be the ServerType that would be returned by an invocation of the GetServerType routine with *SH* as the ServerHandle parameter.

10) Let *SV* be the ServerVersion that would be returned by an invocation of the GetServerVersion routine with *SH* as the ServerHandle parameter.

11) Let *E* be the FDW-environment identified by WrapperEnvHandle.

12) The diagnostics area associated with *E* is emptied.

13) If the maximum number of FS-connections that can be allocated at one time has already been reached, then FSConnectionHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — limit on number of handles exceeded*.

14) Case:

a) If the memory requirements to manage an FS-connection cannot be satisfied, then FSConnectionHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error*.

b) If the resources to manage an FS-connection cannot be allocated for implementation-defined reasons, then FSConnectionHandle is set to zero and an implementation-defined exception condition is raised.

c) Otherwise, the resources to manage an FS-connection are allocated and are referred to as an allocated FS-connection. The allocated FS-connection is assigned a unique value that is returned in FSConnectionHandle.

15) Case:

a) If a connection to *FS* cannot be made, then an exception condition is raised: *foreign-data wrapper-specific condition — unable to establish connection to foreign server*.

b) Otherwise, the connection to *FS* is established.

Conformance Rules

None.

22.7 FreeDescriptor

Function

Release resources associated with a descriptor area.

Definition

```
FreeDescriptor (
      FSConnectionHandle IN      INTEGER,
      DescriptorHandle   IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle and let *HD* be the value of DescriptorHandle.

2) If *FSCH* does not identify an allocated FS-connection, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*; otherwise, let *C* be the allocated FS-connection identified by *FSCH*.

3) The diagnostics area associated with *C* is emptied.

4) If *DH* does not identify an allocated SQL/MED descriptor area associated with C, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

5) Let *D* be the allocated SQL/MED descriptor area identified by *DH*.

6) *D* is deallocated and all its resources are freed.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.8 FreeExecutionHandle

Function

Deallocate an FDW-execution.

Definition

```
FreeExecutionHandle (
      ExecutionHandle  IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *E* be the FDW-execution identified by *EH*.

4) The diagnostics area associated with *E* is emptied.

5) If there is an open cursor associated with *E*, then:

   a) The open cursor associated with *E* is placed in the closed state and its copy of the select source is destroyed.

   b) Any fetched row associated with *E* is removed from association with *E*.

6) The automatically allocated MED descriptor areas associated with *E* are deallocated and all their resources are freed.

7) *E* is deallocated and all its resources are freed.

Conformance Rules

The following restrictions apply for Core SQL:

None.

- 1 Subclause deleted.

22.9 FreeFSConnection

Function

Deallocate a FS-connection.

Definition

```
FreeFSConnection (
      FSConnectionHandle     IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle.

2) If *FSCH* does not identify an allocated FS-connection, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *C* be the allocated FS-connection identified by *FSCH*.

4) The diagnostics area associated with *C* is emptied.

5) If an allocated reply description or FDW-execution is associated with *C*, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error.*

6) *C* is deallocated and all its resources are freed.

Conformance Rules

None.

22.10 FreeReplyHandle

Function

Deallocate an FDW-reply.

Definition

```
FreeReplyHandle (
      ReplyHandle            IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of ReplyHandle.

2) If *RH* does not identify an allocated reply description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *R* be the FDW-reply identified by *RH*.

4) The diagnostics area associated with *R* is emptied.

5) *R* is deallocated and all its resources are freed.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.11 FreeWrapperEnv

Function

Deallocate a FDW-environment.

Definition

```
FreeWrapperEnv (
      WrapperEnvHandle      IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *WEH* be the value of WrapperEnvHandle.

2) If *WEH* does not identify an allocated FDW-environment, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *E* be the allocated FDW-environment identified by *WEH*.

4) The diagnostics area associated with *E* is emptied.

5) If an allocated FS-connection is associated with *E*, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error.*

6) *E* is deallocated and all its resources are freed.

Conformance Rules

None.

22.12 GetAuthorizationId

Function

Get the authorization identifier associated with a user mapping.

Definition

```
GetAuthorizationId (
      UserHandle        IN      INTEGER,
      AuthorizationId   OUT     CHARACTER(L),
      BufferLength      IN      SMALLINT,
      StringLength      OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined length of an <identifier>.

General Rules

1) Let *UH* be the value of UserHandle.

2) If *UH* does not identify an allocated user mapping description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *AID* be the authorization identifier associated with *UH*.

4) Let *BL* be the value of BufferLength.

5) The General Rules of Subclause 21.8, "Character string retrieval", are applied to AuthorizationId, *AID*, *BL*, StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.13 GetDataColumnDetails

Function

Get a details of a column of a row.

Definition

```
GetDataColumnDetails (
      RowHandle         IN      INTEGER,
      ColumnNumber      IN      INTEGER,
      ColumnType        OUT     INTEGER,
      ColumnLength      OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

To be supplied

Conformance Rules

None.

22.14 GetDataColumnValue

Function

Get a column value of a row.

Definition

```
GetDataColumnValue (
      RowHandle         IN      INTEGER,
      ColumnNumber      IN      INTEGER
      Buffer            INOUT   ANY)
      RETURNS SMALLINT
```

General Rules

To be supplied

Conformance Rules

None.

22.15 GetDataRow

Function

Get a row from a data handle.

Definition

```
GetDataRow (
      DataHandle      IN      INTEGER,
      RowNumber       IN      INTEGER,
      RowHandle       OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *DH* be the value of DataHandle.

2) If *DH* does not identify an allocated data row description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *RN* be the value of RowNumber.

4) If *RN* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

5) Let *N* be the number of rows represented by *DH*.

6) If *RN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

7) Information from the *RN*-th row represented by *DH* is retrieved.

   a) Let *RH* be the data row handle.

   b) RowHandle is set to *RH*.

Conformance Rules

None.

22.16 GetDescriptor

Function

Get a field from an SQL/MED descriptor area.

Definition

```
GetDescriptor (
      DescriptorHandle IN      INTEGER,
      RecordNumber     IN      SMALLINT,
      FieldIdentifier  IN      SMALLINT,
      Value            OUT     ANY,
      BufferLength     IN      INTEGER,
      StringLength     OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *D* be the allocated SQL/MED descriptor area identified by DescriptorHandle and let *N* be the value of the COUNT field of *D*.

2) Let *FI* be the value of FieldIdentifier.

3) If *FI* is not one of the code values in Table 33, "Codes used for SQL/MED descriptor fields", then an exception condition is raised: *foreign-data wrapper-specific condition — invalid descriptor field identifier*.

4) Let *RN* be the value of RecordNumber.

5) Let *TYPE* be the value of the Type column in the row of Table 33, "Codes used for SQL/MED descriptor fields", that contains *FI*.

6) The General Rules of Subclause 21.10, "Deferred parameter check", are applied to *D* as the *DESCRIPTOR AREA*.

7) If *TYPE* is 'ITEM', then:

   a) If *RN* is less than 1 (one), then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

   b) If *RN* is greater than *N*, then a completion condition is raised: *no data*.

8) Let *MBR* be the value of the May Be Retrieved column in the row of Table 35, "Ability to retrieve SQL/MED descriptor fields", that contains *FI* and the column that contains the descriptor type *D*.

9) If *MBR* is 'No', then an exception condition is raised: *foreign-data wrapper-specific condition — invalid descriptor field identifier*.

10) If *FI* indicates a descriptor field whose value is the initially undefined value created when the descriptor was created, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid descriptor field identifier*.

11) Let *IDA* be the item descriptor area of *D* specified by *RN*.

12) If *TYPE* is 'HEADER', then header information from the descriptor area *D* is retrieved as follows.

    Case:

    a) If *FI* indicates COUNT, then the value retrieved is *N*.

    b) If *FI* indicates an implementation-defined descriptor header field, then the value retrieved is the value of the implementation-defined descriptor header field identified by *FI*.

    c) Otherwise, if *FI* indicates a descriptor header field defined in Table 33, "Codes used for SQL/MED descriptor fields", then the value retrieved is the value of the descriptor header field identified by *FI*.

13) If *TYPE* is 'ITEM', then item information from the descriptor area *D* is retrieved as follows:

    Case:

    a) If *FI* indicates an implementation-defined descriptor item field, then the value retrieved is the value of the implementation-defined descriptor item field of *IDA* identified by *FI*.

    b) Otherwise, if *FI* indicates a descriptor item field defined in Table 33, "Codes used for SQL/MED descriptor fields", then the value retrieved is the value of the descriptor header field identified by *FI*.

14) Let *V* be the value retrieved.

15) If *FI* indicates a descriptor field whose row in Table 5, "Fields in SQL/MED descriptor areas", contains a Data Type that is not CHARACTER VARYING, then Value is set to *V* and no further rules of this Subclause are applied.

16) Let *BL* be the value of BufferLength.

17) If *FI* indicates a descriptor field whose row in Table 5, "Fields in SQL/MED descriptor areas", contains a Data Type that is CHARACTER VARYING, then the General Rules of Subclause 21.8, "Character string retrieval", are applied with Value, *V*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.17 GetDiagnostics

Function

Get information from an SQL/MED diagnostics area.

Definition

```
GetDiagnostics (
      HandleType     IN      SMALLINT,
      Handle         IN      INTEGER,
      RecordNumber   IN      SMALLINT,
      DiagIdentifier IN      SMALLINT,
      DiagInfo       OUT     ANY,
      BufferLength   IN      SMALLINT,
      StringLength   OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *HT* be the value of HandleType.

2) If *HT* is not one of the code values in Table 34, "Codes used for SQL/MED handle types", then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Case:

   a) If *HT* indicates EXECUTION HANDLE and Handle does not identify an allocated execution description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

   b) If *HT* indicates FSCONNECTION HANDLE and Handle does not identify an allocated FS-connection, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

   c) If *HT* indicates REPLY HANDLE and Handle does not identify an allocated reply description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

   d) If *HT* indicates REQUEST HANDLE and Handle does not identify an allocated request description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

   e) If *HT* indicates DATA HANDLE and Handle does not identify an allocated data row description, then an exception condition is raised: *foreign-data wrapper-specific condition—invalid handle.*

   f) If *HT* indicates SERVER HANDLE and Handle does not identify an allocated foreign-server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

   g) If *HT* indicates TABLEREFERENCE HANDLE and Handle does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

h) If *HT* indicates USER HANDLE and Handle does not identify an allocated user mapping description, then an exception condition is raised: *foreign-data wrapper-specific condition— invalid handle.*

i) If *HT* indicates VALUEEXPRESSION HANDLE and Handle does not identify an allocated value expression description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

j) If *HT* indicates WRAPPER HANDLE and Handle does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

k) If *HT* indicates WRAPPERENV HANDLE and Handle does not identify an allocated FDW-environment, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

4) Let *DI* be the value of DiagIdentifier.

5) If *DI* is not one of the code values in Table 32, "Codes used for SQL/MED diagnostic fields", then an exception condition is raised: *foreign-data wrapper-specific condition — invalid attribute value.*

6) Let *TYPE* be the value of the Type column in the row that contains *DI* in Table 32, "Codes used for SQL/MED diagnostic fields"

7) Let *RN* be the value of RecordNumber.

8) Let *R* be the most recently executed MED routine, other than GetDiagnostics(), for which Handle was passed as the value of an input handle and let *N* be the number of status records generated by the execution of *R*.

NOTE 34 – The GetDiagnostics() routine may cause exception or completion conditions to be raised, but it does not cause diagnostic information to be generated.

9) If TYPE is 'STATUS', then:

a) If *RN* is less than 1 (one), then an exception condition is raised: *invalid condition number.*

b) If *RN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

10) If *TYPE* is 'HEADER', then header information from the diagnostics area associated with the resource identified by Handle is retrieved.

a) If *DI* indicates NUMBER, then the value retrieved is *N*.

b) If *DI* indicates RETURNCODE, then the value retrieved is the code indicating the basic result of the execution of *R*. Subclause 4.14.3, "Return codes", specifies the code values and their meanings.

NOTE 35 – The value retrieved will never indicate **Invalid handle** or **Data needed**, since no diagnostic information is generated if this is the basic result of the execution of *R*.

c) If *DI* indicates MORE, then the value retrieved is

Case:

i) If more conditions were raised during execution of *R* than have been stored in the diagnostics area, then 1 (one).

ii) If all the conditions that were raised during execution of *R* have been stored in the diagnostics area, then 0 (zero).

d) If *DI* indicates an implementation-defined diagnostics header field, then the value retrieved is the value of the implementation-defined diagnostics header field.

11) If *TYPE* is 'STATUS', then information from the *RN*-th status record in the diagnostics area associated with the resource identified by Handle is retrieved.

a) If *DI* indicates SQLSTATE, then the value retrieved is the SQLSTATE value corresponding to the status condition.

b) If *DI* indicates NATIVE_CODE, then the value retrieved is the implementation-defined native error code corresponding to the status condition.

c) If *DI* indicates MESSAGE_TEXT, then the value retrieved is an implementation-defined character string.

NOTE 36 – An implementation may provide <space>s or a zero-length string or a character string that describes the status condition.

d) If *DI* indicates MESSAGE_LENGTH, then the value retrieved is the length in characters of the character string value of MESSAGE_TEXT corresponding to the status condition.

e) If *DI* indicates MESSAGE_OCTET_LENGTH, then the value retrieved is the length in octets of the character string value of MESSAGE_TEXT corresponding to the status condition.

f) If *DI* indicates CLASS_ORIGIN, then the value retrieved is the identification of the naming authority that defined the class value of the SQLSTATE value corresponding to the status condition. That value shall be 'ISO 9075' if the class value is fully defined in Subclause 22.1, "SQLSTATE", in ISO/IEC 9075-2 or Subclause 21.11, "MED-specific status codes", and shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined class value.

g) If *DI* indicates SUBCLASS_ORIGIN, then the value retrieved is the identification of the naming authority that defined the subclass value of the SQLSTATE value corresponding to the status condition. That value shall be 'ISO 9075' if the class value is fully defined in Subclause 22.1, "SQLSTATE", in ISO/IEC 9075-2 or Subclause 21.11, "MED-specific status codes", and shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined subclass value.

h) If *DI* indicates an implementation-defined diagnostics status field, then the value retrieved is the value of the implementation-defined diagnostics status field.

12) Let *V* be the value retrieved.

13) If *DI* indicates a diagnostics field whose row in Table 4, "Fields used in SQL/MED diagnostics areas", contains a Data Type that is neither CHARACTER nor CHARACTER VARYING, then DiagInfo is set to *V* and no further rules of this Subclause are applied.

14) Let *BL* be the value of BufferLength.

15) If *BL* is not greater than zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

16) Let *L* be the length in octets of *V*.

17) If StringLength is not a null pointer, then StringLength is set to *L*.

18) Case:

a) If null termination is false for the SQL-server, then:

i) If *L* is not greater than *BL*, then the first *L* octets of DiagInfo are set to *V* and the values of the remaining octets of DiagInfo are implementation-dependent.

ii) Otherwise, DiagInfo is set to the first *BL* octets of *V*.

b) Otherwise, let *k* be the number of octets in a null terminator in the character set of DiagInfo and let the phrase "implementation-defined null character that terminates a C character string" imply *k* octets, all of whose bits are zero.

i) If *L* is not greater than (*BL*−*k*), then the first (*L*+*k*) octets of DiagInfo are set to *V* concatenated with a single implementation-defined null character that terminates a C character string. The values of the remaining characters of DiagInfo are implementation-dependent.

ii) Otherwise, DiagInfo is set to the first (*BL*−*k*) octets of *V* concatenated with a single implementation-defined null character that terminates a C character string.

Conformance Rules

None.

22.18 GetForeignTableColumn

Function

Get the name and data type of a column of a foreign table.

Definition

```
GetForeignTableColumn (
      TableReferenceHandle IN     INTEGER,
      ColumnNumber         IN     INTEGER,
      ColumnName           OUT    CHARACTER(L1),
      BufferLength1        IN     SMALLINT,
      StringLength1        OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L1* is the value of BufferLength1 and has a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *CN* be the value of ColumnNumber.

4) Let *N* be the number of columns of the foreign table referenced by *TRH*.

5) If *CN* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid column number*.

6) If *CN* is greater than *N*, then a completion condition is raised: *no data* and no further rules of this Subclause are applied.

7) Information from the *CN*-th column is retrieved.

   a) Let *NAME* be the name of the column.

   b) The General Rules of Subclause 21.8, "Character string retrieval", are applied to ColumnName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.19 GetForeignTableColumnOption

Function

Get the name and value of a generic option associated with a column of a foreign table, given an option number.

Definition

```
GetForeignTableColumnOption (
      TableReferenceHandle IN     INTEGER,
      ColumnName           IN     CHARACTER(L),
      NameLength           IN     SMALLINT,
      OptionNumber         IN     INTEGER,
      OptionName           OUT    CHARACTER(L1),
      BufferLength1        IN     SMALLINT,
      StringLength1        OUT    SMALLINT,
      OptionValue          OUT    CHARACTER(L2),
      BufferLength2        IN     SMALLINT,
      StringLength2        OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L*, *L1*, and *L2* are the value of NameLength, BufferLength1, and BufferLength2, respectively, and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition 0 invalid handle.*

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of ColumnName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of ColumnName and let *NO* be the number of octets occupied by those *N* characters. If NO $\neq$ *L*, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid column name.*

ii) Otherwise, let *CN* be the first *L* octets of ColumnName and let *TCN* be the value of

```
TRIM ( BOTH ' ' FROM CN )
```

6) Let *NC* be the number of columns of the foreign table referenced by *TRH*.

7) Case:

a) If *TCN* is not equivalent to the name of a column of the foreign table referenced by (TRH), then an exception condition is raised: *foreign-data wrapper-specific condition — column name not found*.

b) Otherwise:

i) Let *ON* be the value of OptionNumber.

ii) Let *N* be the number of generic options associated with the column identified by *TCN*.

iii) If *ON* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

iv) If *ON* is greater than *N*, then a completion condition is raised: *no data* and no further rules of this Subclause are applied.

v) Information from the *ON*-th generic option associated with the column identified by TCN is retrieved.

1) Let *NAME* be the name of the generic option.

2) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

3) Let *VALUE* be the value of the generic option.

4) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.20 GetForeignTableColumnOptionValueByName

Function

Get the value of a generic option associated with a column of a foreign table, given the option name.

Definition

```
GetForeignTableColumnOptionValueByName (
      TableReferenceHandle IN     INTEGER,
      ColumnName           IN     CHARACTER(L),
      NameLength           IN     SMALLINT,
      OptionName           IN     CHARACTER(L1),
      BufferLength1        IN     SMALLINT,
      OptionValue          OUT    CHARACTER(L2),
      BufferLength2        IN     SMALLINT,
      StringLength2        OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L*, *L1*, and *L2* are the value of NameLength, BufferLength1, and BufferLength2, respectively, and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of ColumnName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of ColumnName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid column name.*

ii) Otherwise, let *CN* be the first *L* octets of ColumnName and let *TCN* be the value of

```
TRIM ( BOTH ' ' FROM CN )
```

6) Let *NC* be the number of columns of the foreign table referenced by *TRH*.

7) Case:

a) If *TCN* is not equivalent to the name of a column of the foreign table referenced by TRH, then an exception condition is raised: *foreign-data wrapper-specific condition — column name not found*.

b) Otherwise:

i) Let *BL* be the value of BufferLength1.

ii) Case:

1) If *BL* is not negative, then let *BL1* be *BL*.

2) If *BL* indicates NULL TERMINATED, then let *BL1* be the number of octets of OptionName that precede the implementation-defined null character that terminates a C character string.

3) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

iii) Case:

1) If *BL1* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

2) Otherwise:

A) Let *BN* be the number of whole characters in the first *BL1* octets of OptionName and let *BNO* be the number of octets occupied by those *BN* characters. If *BNO* ≠ *BL1*, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option name.*

B) Otherwise, let *ON* be the first *BL1* octets of OptionName and let *TON* be the value of

```
TRIM ( BOTH ' ' FROM ON )
```

iv) Let *N* be the number of generic options associated with *TRH*.

v) Case:

1) If *TON* is not equivalent to the name of a generic option associated with *TRH*, then an exception condition is raised: *foreign-data wrapper-specific condition — option name not found*.

2) Otherwise:

A) Let *VALUE* be the value of the generic option associated with *TRH* whose name is *TON*.

B) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.21 GetForeignTableColumnType

Function

Get the data type of a column of a foreign table.

Definition

```
GetForeignTableColumnType (
      TableReferenceHandle IN     INTEGER,
      ColumnName           IN     CHARACTER(L1),
      NameLength           IN     SMALLINT,
      ColumnType           OUT    INTEGER )
      RETURNS SMALLINT
```

where *L1* is the value of NameLength and has a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of ColumnName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of ColumnName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid column name.*

      ii) Otherwise, let *CN* be the first *L* octets of ColumnName and let *TCN* be the value of

```
TRIM ( BOTH ' ' FROM CN )
```

6) Let *NC* be the number of columns of the foreign table referenced by *TRH*.

7) Case:

   a) If *TCN* is not equivalent to the name of a column of the foreign table referenced by *TRH*, then an exception condition is raised: *foreign-data wrapper-specific condition — column name not found*.

   b) Otherwise:

      i) Let *DT* be the data type of the column of the foreign table referenced by *TRH* whose name is *TCN*.

      ii) ColumnType is set to *DT*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.22 GetForeignTableOption

Function

Get the name and value of a generic option associated with a foreign table, given an option number.

Definition

```
GetForeignTableOption (
      TableReferenceHandle IN     INTEGER,
      OptionNumber         IN     INTEGER,
      OptionName           OUT    CHARACTER(L1),
      BufferLength1        IN     SMALLINT,
      StringLength1        OUT    SMALLINT,
      OptionValue          OUT    CHARACTER(L2),
      BufferLength2        IN     SMALLINT,
      StringLength2        OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *ON* be the value of OptionNumber.

4) Let *N* be the number of generic options associated with *TRH*.

5) If *ON* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

6) If *ON* is greater than *N*, then a completion condition is raised: *no data* and no further rules of this Subclause are applied.

7) Information from the *ON*-th generic option associated with *TRH* is retrieved.

   a) Let *NAME* be the name of the generic option.

   b) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *VALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.23 GetForeignTableOptionValueByName

Function

Get the value of a generic option associated with a foreign table, given the option name.

Definition

```
GetForeignTableOptionValueByName (
      TableReferenceHandle IN     INTEGER,
      OptionName           IN     CHARACTER(L1),
      BufferLength1        IN     SMALLINT,
      OptionValue          OUT    CHARACTER(L2),
      BufferLength2        IN     SMALLINT,
      StringLength2        OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *NL* be the value of BufferLength1.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of OptionName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of OptionName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option name.*

      ii) Otherwise, let *ON* be the first *L* octets of OptionName and let *TON* be the value of

```
TRIM ( BOTH ' ' FROM ON )
```

6) Let *N* be the number of generic options associated with *TRH*.

7) Case:

   a) If *TON* is not equivalent to the name of a generic option associated with *TRH*, then an exception condition is raised: *foreign-data wrapper-specific condition — option name not found*.

   b) Otherwise:

      i) Let *VALUE* be the value of the generic option associated with *TRH* whose name is *TON*.

      ii) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.24 GetForeignTableServerName

Function

Get the name of the foreign server associated with a foreign table.

Definition

```
GetForeignTableServerName (
      TableReferenceHandle IN     INTEGER,
      ServerName           OUT    CHARACTER(L),
      BufferLength         IN     SMALLINT,
      StringLength         OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined length of an <identifier>.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *SN* be the server name associated with the foreign table identified by *TRH*.

4) Let *BL* be the value of BufferLength.

5) The General Rules of Subclause 21.8, "Character string retrieval", are applied to ServerName, *SN*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.25 GetNumberOfDataColumns

Function

Get the number of columns described by a data handle.

Definition

```
GetNumberOfDataColumns (
      DataHandle          IN      INTEGER,
      NumberofColumns     OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *DH* be the value of DataHandle.

2) If *DH* does not identify an allocated data row description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *N* be the number of columns described by *DH*.

4) NumberOfColumns is set to *N*.

Conformance Rules

None.

22.26 GetNumberOfDataRows

Function

Get the number of data rows described by a data handle.

Definition

```
GetNumberOfDataRows (
      DataHandle          IN      INTEGER,
      NumberofRows        OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *DH* be the value of DataHandle.

2) If *DH* does not identify an allocated data row description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *N* be the number of rows described by *DH*.

4) NumberOfRows is set to *N*.

Conformance Rules

None.

22.27 GetNumberOfForeignTableColumns

Function

Get the number of columns of a foreign table.

Definition

```
GetNumberOfForeignTableColumns (
      TableReferenceHandle            IN      INTEGER,
      NumberOfForeignTableColumns     OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *N* be the degree of the foreign table referenced by *TRH*.

4) NumberOfForeignTableColumns is set to *N*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.28 GetNumberOfForeignTableColumnOptions

Function

Get the number of generic options of a column of a foreign table.

Definition

```
GetNumberOfForeignTableColumnOptions (
      TableReferenceHandle           IN      INTEGER,
      ColumnName                     IN      CHARACTER(L),
      NameLength                     IN      SMALLINT,
      OptionCount                    OUT     INTEGER )
      RETURNS SMALLINT
```

where *L* is the value of NameLength and has a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of ColumnName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of ColumnName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid column name.*

      ii) Otherwise, let *CN* be the first *L* octets of ColumnName and let *TCN* be the value of

```
TRIM ( BOTH ' ' FROM CN )
```

6) Let *NC* be the number of columns of the foreign table referenced by *TRH*.

7) Case:

   a) If *TCN* is not equivalent to the name of a column of the foreign table referenced by TRH, then an exception condition is raised: *foreign-data wrapper-specific condition — column name not found*.

   b) Otherwise:

      i) Let *ON* be the number of generic options of the column of the foreign table referenced by *TRH* whose name is *TCN*.

      ii) OptionCount is set to *ON*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.29 GetNumberOfForeignTableOptions

Function

Get the number of generic options of a foreign table.

Definition

```
GetNumberOfForeignTableOptions (
      TableReferenceHandle            IN      INTEGER,
      OptionCount                     OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *N* be the number of generic options of the foreign table referenced by *TRH*.

4) OptionCount is set to *N*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.30 GetNumberOfReplySelectListElements

Function

Get the number of <value expressions>s in the <select list> of a query that the foreign-data wrapper is capable of handling.

Definition

```
GetNumberOfReplySelectListElements (
      ReplyHandle                    IN      INTEGER,
      NumberOfSelectListElements     OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of ReplyHandle.

2) If *RH* does not identify an allocated reply description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <value expression> elements in the <select list> of *Q* that the foreign-data wrapper is capable of handling.

5) NumberOfSelectListElements is set to *N*.

Conformance Rules

None.

22.31 GetNumberOfReplyTableReferences

Function

Get the number of <table reference>s in the <from clause> of a query that can be processed by the foreign-data wrapper.

Definition

```
GetNumberOfReplyTableReferences (
      ReplyHandle                IN      INTEGER,
      NumberOfTableReferences    OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of ReplyHandle.

2) If *RH* does not identify an allocated reply description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <table reference> elements in the <from clause> of *Q* that the foreign-data wrapper is capable of handling.

5) NumberOfTableReferences is set to *N*.

Conformance Rules

None.

22.32 GetNumberOfSelectListElements

Function

Get the number of <value expressions>s in the <select list> of a query.

Definition

```
GetNumberOfSelectListElements (
      RequestHandle                  IN      INTEGER,
      NumberOfSelectListElements     OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of RequestHandle.

2) If *RH* does not identify an allocated request description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <value expression> elements in the <select list> of *Q*.

5) NumberOfSelectListElements is set to *N*.

Conformance Rules

None.

22.33 GetNumberOfServerOptions

Function

Get the number of generic options associated with the foreign server.

Definition

```
GetNumberOfServerOptions (
      ServerHandle           IN      INTEGER,
      OptionCount            OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *SH* be the value of ServerHandle.

2) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) OptionCount is set to the number of generic options associated with *SH*.

Conformance Rules

None.

22.34 GetNumberOfTableReferenceElements

Function

Get the number of <table reference>s contained in the <from clause> of a query.

Definition

```
GetNumberOfTableReferenceElements (
      RequestHandle              IN      INTEGER,
      NumberOfTableRefernces     OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of RequestHandle.

2) If *RH* does not identify an allocated request description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <table reference> elements in the <from clause> of *Q*.

5) NumberOfTableReferenceElements is set to *N*.

Conformance Rules

None.

22.35 GetNumberOfUserOptions

Function

Get the number of generic options of a user mapping.

Definition

```
GetNumberOfUserOptions (
      UserHandle                  IN      INTEGER,
      OptionCount                 OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *UH* be the value of UserHandle.

2) If *UH* does not identify an allocated user mapping description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *N* be the number of generic options of the user mapping described by *UH*.

4) OptionCount is set to *N*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.36 GetNumberOfWrapperOptions

Function

Get the number of generic options of a foreign-data wrapper.

Definition

```
GetNumberOfWrapperOptions (
      WrapperHandle               IN      INTEGER,
      OptionCount                 OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *WH* be the value of WrapperHandle.

2) If *WH* does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *N* be the number of generic options of the foreign-data wrapper described by *WH*.

4) OptionCount is set to *N*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.37 GetOptions

Function

Request the foreign-data wrapper to supply information about the capabilities and other information of the requested object.

Definition

```
GetOptions (
      InputHandle         IN      INTEGER,
      HandleType          IN      SMALLINT,
      OptionsObject       IN      CHARACTER VARYING(L1),
      OptionsXML          OUT     CHARACTER VARYING(L2),
      OptionsBufferLength IN      INTEGER,
      OptionsOutputLength OUT     INTEGER )
      RETURNS SMALLINT
```

where: *L1* is the implementation-defined maximum length of a variable-length character string, and *L2* is determined by the value of OptionsOutputLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

General Rules

1) Let *IH* be the value of InputHandle and let *HT* be the value of HandleType.

2) If *HT* is not one of the code values in Table 34, "Codes used for SQL/MED handle types", then an exception condition is raised: *foreign-data wrapper-specific exception — invalid handle.*

3) If *IH* does not identify a handle of the type indicated by *HT*, then an exception condition is raised: *foreign-data wrapper-specific exception — invalid handle.*

4) Case:

   a) If *HT* indicates WRAPPER HANDLE, then

      Case:

      i) If the foreign-data wrapper *FDW* described by *IH* cannot return a report of its capabilities and other information, then a completion condition is raised: *no data*.

      ii) Otherwise, an XML document *XD* is created that describes the capabilities of *FDW*.

   b) If *HT* indicates SERVER HANDLE, then

      Case:

      i) If the foreign server *FS* described by *IH* cannot return a report of its capabilities and other information, then a completion condition is raised: *no data*.

      ii) Otherwise, an XML document *XD* is created that describes the capabilities of *FS*.

5) Case:

   a) If the length in octets *LOXD* of *XD* is greater than the value of OptionsBufferLength, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

b) Otherwise, the value of OptionsOutputLength is set to *LOXD* and the value of OptionsXML is set to *XD*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.38 GetReplySelectListElement

Function

Get the number of a <value expression> element from the <select list> of a query that the foreign-data wrapper is capable of handling.

Definition

```
GetReplySelectListElement (
      ReplyHandle                IN      INTEGER,
      Index                      IN      SMALLINT,
      SelectListElementNumber    OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of ReplyHandle.

2) If RH does not identify an allocated reply description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *I* be the value of Index.

4) If *I* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index.*

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <value expression> elements in the <select list> of *Q* that the foreign-data wrapper is capable of handling.

7) If *I* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) Information from the *I*-th <value expression> that the foreign-data wrapper is capable is handling is retrieved.

   a) Let *TRN* be the number of the <value expression> element in the <select list> of *Q*.

   b) SelectListElementNumber is set to *TRN*.

Conformance Rules

None.

22.39 GetReplyTableReference

Function

Get the number of a <table reference> element from the <from clause> of a query that the foreign-data wrapper is capable of handling.

Definition

```
GetReplyTableReference (
      ReplyHandle             IN      INTEGER,
      Index                   IN      SMALLINT,
      TableReferenceNumber    OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of ReplyHandle.

2) If *RH* does not identify an allocated reply description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *I* be the value of Index.

4) If *I* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index.*

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <table reference> elements in the <from clause> of *Q* that the foreign-data wrapper is capable of handling.

7) If *I* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) Information from the *I*-th <table reference> that the foreign-data wrapper is capable of handling is retrieved.

   a) Let *TRN* be the number of the <table reference> element in the <from clause> of *Q*.

   b) TableReferenceNumber is set to *TRN*.

Conformance Rules

None.

22.40 GetSelectListElement

Function

Get a <value expression> element from the <select list> of a query.

Definition

```
GetSelectListElement (
      RequestHandle              IN      INTEGER,
      SelectListElementNumber    IN      SMALLINT,
      ValueExpressionHandle      OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of RequestHandle.

2) If *RH* does not identify an allocated request description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *SLEN* be the value of SelectListElementNumber.

4) If *SLEN* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <value expression> elements in the <select list> of *Q*.

7) If *SLEN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) Information from the *SLEN*-th <value expression> associated with *Q* is retrieved.

   a) Let *VEH* be the value expression handle associated with the <value expression>.

   b) ValueExpressionHandle is set to *VEH*.

Conformance Rules

None.

22.41 GetSelectListElementType

Function

Get the type of a <value expression>

Definition

```
GetSelectListElementType (
      ValueExpressionHandle      IN      INTEGER,
      ValueExpressionType        OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *VEH* be the value of ValueExpressionHandle.

2) If *VEH* does not identify an allocate <value expression> description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *VET* be the type of the <value expression> associated with *VEH*.

4) ValueExpressionType is set to *VET*.

Conformance Rules

None.

22.42 GetServerName

Function

Get the name of the foreign server.

Definition

```
GetServerName (
      ServerHandle        IN      INTEGER,
      ServerName          OUT     CHARACTER(L),
      BufferLength        IN      SMALLINT,
      StringLength        OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value of equal to the implementation-defined length of an <identifier>.

General Rules

1) Let *SH* be the value of ServerHandle.

2) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *SN* be the server name associated with *SH*.

4) Let *BL* be the value of BufferLength.

5) The General Rules of Subclause 21.8, "Character string retrieval", are applied to ServerName, *SN*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.43 GetServerOption

Function

Get the value of a generic option associated with a foreign server.

Definition

```
GetServerOption (
      ServerHandle       IN      INTEGER,
      OptionNumber       IN      INTEGER,
      OptionName         OUT     CHARACTER(L1),
      BufferLength1      IN      SMALLINT,
      StringLength1      OUT     SMALLINT,
      OptionValue        OUT     CHARACTER(L2),
      BufferLength2      IN      SMALLINT,
      StringLength2      OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *SH* be the value of ServerHandle.

2) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *ON* be the value of OptionNumber.

4) Let *N* be the number of generic options associated with *SH*.

5) If *ON* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

6) If *ON* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

7) Information from the *ON*-th generic option associated with the *SH* is retrieved.

   a) Let *NAME* be the name of the generic option.

   b) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *VALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.44 GetServerOptionValueByName

Function

Get the value of a generic option associated with a foreign server.

Definition

```
GetServerOption (
      ServerHandle          IN      INTEGER,
      OptionName            IN      CHARACTER(L1),
      BufferLength          IN      SMALLINT,
      OptionValue           OUT     CHARACTER(L2),
      BufferLength2         IN      SMALLINT,
      StringLength2         OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *SH* be the value of ServerHandle.

2) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *NL* be the value of NameLength1.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of OptionName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of OptionName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option name.*

      ii) Otherwise, let *ON* be the first *L* octets of OptionName and let *TON* be the value of:

          TRIM (BOTH ' ' FROM *ON*)

6) Let *N* be the number of generic options associated with *SH*.

7) Case:

   a) If *TON* is not equivalent to one of the names of one of the *N* generic option associated with *SH*, then an exception condition is raised: *foreign-data wrapper-specific condition — option name not found*.

   b) Otherwise:

      i) Let *VALUE* be the value of the generic option associated with *SH* whose name is *TON*.

      ii) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.45 GetServerType

Function

Get the type of the foreign server.

Definition

```
GetServerType (
      ServerHandle       IN      INTEGER,
      ServerType         OUT     CHARACTER(L),
      BufferLength       IN      SMALLINT,
      StringLength       OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *SH* be the value of ServerHandle.

2) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *ST* be the server type associated with *SH*.

4) Let *BL* be the value of BufferLength.

5) The General Rules of Subclause 21.8, "Character string retrieval", are applied to ServerType, *ST*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.46 GetServerVersion

Function

Get the version of the foreign server.

Definition

```
GetServerVersion (
      ServerHandle          IN      INTEGER,
      ServerVersion         OUT     CHARACTER(L),
      BufferLength          IN      SMALLINT,
      StringLength          OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *SH* be the value of ServerHandle.

2) If *SH* does not identify an allocated foreign server description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *SV* be the server version associated with *SH*.

4) Let *BL* be the value of BufferLength.

5) The General Rules of Subclause 21.8, "Character string retrieval", are applied to ServerVersion, *SV*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.47 GetSPDHandle

Function

Get the descriptor handle of the server parameter descriptor associated with an ExecutionHandle.

Definition

```
GetSPDHandle (
      ExecutionHandle       IN      INTEGER,
      SPDHandle             OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *SPDH* be the descriptor handle of the server parameter descriptor associated with *EH*.

4) SPDHandle is set to *SPDH*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.48 GetSRDHandle

Function

Get the descriptor handle of the server row descriptor associated with an ExecutionHandle.

Definition

```
GetSRDHandle (
      ExecutionHandle       IN      INTEGER,
      SRDHandle             OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *SRDH* be the descriptor handle of the server row descriptor associated with *EH*.

4) SRDHandle is set to *SRDH*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.49 GetTRDHandle

Function

Get the descriptor handle of the table reference descriptor associated with a TableReferenceHandle.

Definition

```
GetTRDHandle (
      TableReferenceHandle  IN      INTEGER,
      TRDHandle             OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *TRDH* be the descriptor handle of the table reference descriptor associated with *TRH*.

4) TRDHandle is set to *TRDH*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.50 GetWRDHandle

Function

Get the descriptor handle of the wrapper row descriptor associated with an ExecutionHandle.

Definition

```
GetWRDHandle (
      ExecutionHandle       IN      INTEGER,
      WRDHandle             OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *WRDH* be the descriptor handle of the wrapper row descriptor associated with *EH*.

4) WRDHandle is set to *WRDH*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.51 GetWPDHandle

Function

Get the descriptor handle of the wrapper parameter descriptor associated with an ExecutionHandle.

Definition

```
GetWPDHandle ( ExecutionHandle       IN      INTEGER, WPDHandle            OUT     INTEGER )
       RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *WPDH* be the descriptor handle of the wrapper parameter descriptor associated with *EH*.

4) WPDHandle is set to *WPDH*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.52 GetTableReferenceElement

Function

Get a <table reference> element from the <from clause> of a query.

Definition

```
GetTableReferenceElement (
      RequestHandle                 IN      INTEGER,
      TableRefer nceElementNumber   IN      INTEGER,
      TableReferenceHandle          OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of RequestHandle.

2) If *RH* does not identify an allocated request description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *TREN* be the value of TableReferenceElementNumber.

4) If *TREN* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <table reference> elements in the <from clause> of *Q*.

7) If *TREN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) Information from the *TREN*-th <table reference> associated with *Q* is retrieved.

   a) Let *TRH* be the table reference handle associated with the <table reference>.

   b) TableReferenceHandle is set to *TRH*.

Conformance Rules

None.

22.53 GetTableReferenceElementType

Function

Get the type of a <table reference>.

Definition

```
GetTableReferenceElementType (
      TableReferenceHandle      IN      INTEGER,
      TableReferenceType        OUT     SMALLINT )
      RETURNS SMALLINT
```

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *TRT* be the type of the <table reference> associated with *TRH*.

4) TableReferenceType is set to *TRT*.

Conformance Rules

None.

22.54 GetTableReferenceTableName

Function

Get the table name of a TABLE_NAME <table reference>.

Definition

```
GetTableReferenceElementType (
      TableReferenceHandle      IN      INTEGER,
      TableName                 OUT     CHARACTER(L),
      BufferLength              IN      SMALLINT,
      StringLength              OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *TRH* be the value of TableReferenceHandle.

2) If *TRH* does not identify an allocated table reference description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) If *TRH* does not identify a <table reference> with a type of TABLE_NAME, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

4) Let *NAME* be the value of the table name of the <table reference>.

5) Let *BL* be the value of BufferLength.

6) The General Rules of Subclause 21.8, “Character string retrieval”, are applied to TableName, *NAME*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.55 GetUserOption

Function

Get the name and value of a generic option associated with a user mapping, given an option number.

Definition

```
GetUserOption (
      UserHandle                IN      INTEGER,
      OptionNumber              IN      INTEGER,
      OptionName                OUT     CHARACTER(L1),
      BufferLength1             IN      SMALLINT,
      StringLength1             OUT     SMALLINT,
      OptionValue               OUT     CHARACTER(L2),
      BufferLength2             IN      SMALLINT,
      StringLength2             OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *UH* be the value of UserHandle.

2) If *UH* does not identify an allocated user mapping description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *ON* be the value of OptionNumber.

4) Let *N* be the number of generic options associated with *UH*.

5) If *ON* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

6) If *ON* is greater than *N*, then a completion condition is raised: *no data* and no further rules of this Subclause are applied.

7) Information from the *ON*-th generic option associated with *UH* is retrieved.

   a) Let *NAME* be the name of the generic option.

   b) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *VALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.56 GetUserOptionValueByName

Function

Get the value of a generic option associated with a user mapping, given the option name.

Definition

```
GetUserOptionValueByName (
      UserHandle                IN      INTEGER,
      OptionName                IN      CHARACTER(L1),
      BufferLength1             IN      SMALLINT,
      OptionValue               OUT     CHARACTER(L2),
      BufferLength2             IN      SMALLINT,
      StringLength2             OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *UH* be the value of UserHandle.

2) If *UH* does not identify an allocated user mapping description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *NL* be the value of BufferLength1.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of OptionName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of OptionName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option name.*

      ii) Otherwise, let *ON* be the first *L* octets of OptionName and let *TON* be the value of

```
TRIM ( BOTH ' ' FROM ON )
```

6) Let *N* be the number of generic options associated with *UH*.

7) Case:

   a) If *TON* is not equivalent to the name of a generic option associated with *UH*, then an exception condition is raised: *foreign-data wrapper-specific condition — option name not found*.

   b) Otherwise:

      i) Let *VALUE* be the value of the generic option associated with *UH* whose name is *TON*.

      ii) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.57 GetValueExpressionColumnName

Function

Get the column name of a COLUMN_NAME <value expression>

Definition

```
GetValueExpressionColumnName (
      ValueExpressionHandle      IN      INTEGER,
      ColumnName                 OUT     CHARACTER(L),
      BufferLength               IN      SMALLINT,
      StringLength               OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value of equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *VEH* be the value of ValueExpressionHandle.

2) If *VEH* does not identify an allocated value expression description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) If *VEH* does not identify a <value expression> with a type of COLUMN_NAME, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

4) Let *NAME* be the value of the column name of the <value expression>.

5) Let *BL* be the value of BufferLength.

6) The General Rules of Subclause 21.8, "Character string retrieval", are applied to ColumnName, *NAME*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

None.

22.58 GetWrapperLibraryName

Function

Get the library name associated with a foreign-data wrapper.

Definition

```
GetWrapperLibraryName (
      WrapperHandle           IN      INTEGER,
      WrapperLibraryName      OUT     CHARACTER(L),
      BufferLength            IN      SMALLINT,
      StringLength            OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

General Rules

1) Let *WH* be the value of WrapperHandle.

2) If *WH* does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *WL* be the name of the library included in the foreign-data wrapper descriptor of the foreign-data wrapper associated with *WH*.

4) The General Rules of Subclause 21.8, "Character string retrieval", are applied to WrapperLibraryName, *WL*, BufferLength, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.59 GetWrapperName

Function

Get the name of a foreign-data wrapper.

Definition

```
GetWrapperName (
      WrapperHandle           IN      INTEGER,
      WrapperName             OUT     CHARACTER(L),
      BufferLength            IN      SMALLINT,
      StringLength            OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to (2n+1) where n is the implementation-defined length of an <identifier>.

General Rules

1) Let *WH* be the value of WrapperHandle.

2) If *WH* does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *WN* be the foreign-data wrapper name included in the foreign-data wrapper descriptor of the foreign-data wrapper associated with *WH*.

4) The General Rules of Subclause 21.8, "Character string retrieval", are applied to WrapperName, *WN*, BufferLength, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.60 GetWrapperOption

Function

Get the name and value of a generic option associated with a foreign-data wrapper, given an option number.

Definition

```
GetWrapperOption (
      WrapperHandle          IN     INTEGER,
      OptionNumber           IN     INTEGER,
      OptionName             OUT    CHARACTER(L1),
      BufferLength1          IN     SMALLINT,
      StringLength1          OUT    SMALLINT,
      OptionValue            OUT    CHARACTER(L2),
      BufferLength2          IN     SMALLINT,
      StringLength2          OUT    SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *WH* be the value of WrapperHandle.

2) If *WH* does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *ON* be the value of OptionNumber.

4) Let *N* be the number of generic options associated with *WH*.

5) If *ON* is less than 1 (one), then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option index*.

6) If *ON* is greater than *N*, then a completion condition is raised: *no data* and no further rules of this Subclause are applied.

7) Information from the *ON*-th generic option associated with *WH* is retrieved.

   a) Let *NAME* be the name of the generic option.

   b) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *VALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.61 GetWrapperOptionValueByName

Function

Get the value of a generic option associated with a foreign-data wrapper, given the option name.

Definition

```
GetWrapperOptionValueByName (
      WrapperHandle           IN      INTEGER,
      OptionName              IN      CHARACTER(L1),
      BufferLength1           IN      SMALLINT,
      OptionValue             OUT     CHARACTER(L2),
      BufferLength2           IN      SMALLINT,
      StringLength2           OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L1* and *L2* are the value of BufferLength1 and BufferLength2 and have a maximum value equal to the implementation-defined length of a variable-length character string.

General Rules

1) Let *WH* be the value of WrapperHandle.

2) If *WH* does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.* Let *NL* be the value of BufferLength1.

3) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of OptionName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

4) Case:

   a) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise:

      i) Let *N* be the number of whole characters in the first *L* octets of OptionName and let *NO* be the number of octets occupied by those *N* characters. If $NO \neq L$, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid option name.*

      ii) Otherwise, let *ON* be the first *L* octets of OptionName and let *TON* be the value of

```
TRIM ( BOTH ' ' FROM ON )
```

5) Let *N* be the number of generic options associated with *WH*.

6) Case:

   a) If *TON* is not equivalent to the name of a generic option associated with *WH*, then an exception condition is raised: *foreign-data wrapper-specific condition — option name not found*.

   b) Otherwise:

      i) Let *VALUE* be the value of the generic option associated with *WH* whose name is *TON*.

      ii) The General Rules of Subclause 21.8, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.62 InitForeignRequest

Function

Determine whether a foreign-data wrapper can execute an SQL-statement.

Definition

```
InitForeignRequest (
      FSConnectionHandle      IN      INTEGER,
      RequestHandle           IN      INTEGER,
      ReplyHandle             OUT     INTEGER,
      ExecutionHandle         OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle.

2) If *FSCH* does not identify an allocated foreign server connection, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) If an exception condition is raised in any of the routines invoked in any of the following General Rules, then the diagnostics records returned by the invoked routines are transferred to the diagnostics area associated with the FSConnectionHandle and further processing of this routine is terminated as if the exception condition had been raised in this routine.

4) Let *RH* be the value of RequestHandle.

5) Let *NTRE* be the NumberOfTableReferenceElements returned by an invocation of the GetNumberofTableReferenceElements routine with *RH* as the RequestHandle parameter.

6) Let TRH_i be the TableReferenceHandle returned by invocation of the GetTableReferenceElement routine with *RH* as the RequestHandle parameter and *i* as the TableReferenceElementNumber parameter for 1 (one) $\leq i \leq NTRE$.

7) Let $TRDH_i$ be the TableReferenceDescriptorHandle that would be returned by invocation of the GetTRDHandle( ) routine with TRH_i as the TableReferenceHandle parameter.

8) Let NC_i be the value of the COUNT descriptor field that would be returned by invocation of the GetDescriptor( ) routine with $TRDH_i$ as the DescriptorHandle parameter, 0 (zero) as the RecordNumber parameter, and the code for COUNT from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

9) Let DT_{ij} be the effective data type of the *j*-th column, for 1 (one) $\leq j \leq NC_i$, as represented by the values of the TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME fields that would be returned by separate invocations of the GetDescriptor( ) routine with $TRDH_i$ as the DescriptorHandle parameter, *j* as the RecordNumber parameter, and the code for the fields TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,

CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME from Table 33, "Codes used for SQL/MED descriptor fields", as the FieldIdentifier parameter.

10) Let TRT_i be the TableReferenceType returned by invocation of the GetTableReferenceElementType routine with TRH_i as the TableReferenceHandle parameter.

11) Let TN_i be the TableName returned by invocation of the GetTableReferenceTableName routine with TRH_i as the TableReferenceHandle parameter.

12) Let *NSLE* be the NumberOfSelectListElements returned by an invocation of the GetNumberofSelectListElements routine with *RH* as the RequestHandle parameter.

13) Let VEH_k be the ValueExpressionHandle that would be returned by invocation of the GetSelectListElement( ) routine with *RH* as the RequestHandle parameter, and *k* as the SelectListElementNumber parameter for 1 (one) $\leq k \leq NSLE$.

14) Let VET_k be the ValueExpressType that would be returned by invocation of the GetSelectListElementType( ) routine with VEH_k as the ValueExpressionHandle parameter.

15) Let CN_k be the ColumnName that would be returned by invocation of the GetValueExpressionColumnName( ) routine with VEH_k as the ValueExpressionHandle parameter.

16) Let *EH* be the allocated FDW-environment associated with *FSCH*.

17) If the maximum number of FDW-replies that can be allocated at one time has already been reached, then ReplyHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — limit on number of handles exceeded*.

18) Case:

    a) If the memory requirements to manage an FDW-reply cannot be satisfied, then ReplyHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error*.

    b) If the resources to manage an FDW-reply cannot be allocated for implementation-defined reasons, then ReplyHandle is set to zero and an implementation-defined exception condition is raised.

    c) Otherwise, the resources to manage an FDW-reply are allocated and are referred to as an *allocated reply description*. The allocated reply description is assigned a unique value that is returned in ReplyHandle.

19) Case:

    a) If the foreign-data wrapper cannot create a FDW-reply that corresponds to *RH* as described by *NTRE*, (TRH_i, $TRDH_i$, NC_i, TRT_i, and TN_i, for 1 (one) $\leq i \leq NTRE$), (DT_{ij}, for 1 (one) $\leq i \leq NTRE$ and 1 (one) $\leq j \leq NC_i$), *NSLE*, and (VEH_k, VET_k, and CN_k, for 1 (one) $\leq k \leq NSLE$), then an exception condition is raised: *foreign-data wrapper-specific condition — unable to create reply*.

    b) Otherwise, the FDW-reply corresponding to *RH* is created.

20) If the maximum number of FDW-executions that can be allocated at one time has already been reached, then ExecutionHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — limit on number of handles exceeded*.

21) Case:

a) If the memory requirements to manage an FDW-execution cannot be satisfied, then ExecutionHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error*.

b) If the resources to manage an FDW-execution cannot be allocated for implementation-defined reasons, then ExecutionHandle is set to zero and an implementation-defined exception condition is raised.

c) Otherwise, the resources to manage an FDW-execution are allocated and are referred to as an *allocated execution description*. The allocated execution description is assigned a unique value that is returned in ExecutionHandle.

22) Case:

a) If the foreign-data wrapper cannot create an FDW-execution that corresponds to *RH* as described by *NTRE*, (TRH_i, $TRDH_i$, NC_i, TRT_i, and TN_i, for 1 (one) $\leq i \leq$ *NTRE*), (DT_{ij}, for 1 (one) $\leq i \leq$ *NTRE* and 1 (one) $\leq j \leq NC_i$), *NSLE*, and (VEH_k, VET_k, and CN_k, for 1 (one) $\leq k \leq$ *NSLE*), then an exception condition is raised: *foreign-data wrapper-specific condition — unable to create execution*.

b) Otherwise, the FDW-execution corresponding to *RH* is created.

23) The following MED descriptor areas are automatically allocated and associated with the allocated FDW-execution:

a) A wrapper parameter descriptor *WPD*.

b) A wrapper row descriptor *WRD*.

c) A server parameter descriptor *SPD*.

d) A server row descriptor *SRD*.

For each of these descriptor areas, fields with non-blank entries in Table 37, "SQL/MED descriptor field default values", are set to the specified default values. All other fields in the SQL/MED item descriptor areas are initially undefined.

24) Let *P* be the SQL-statement represented by the FDW-execution.

25) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation*.

a) *P* does not conform to the Format, Syntax Rules or Access Rules for a <preparable statement>.

b) *P* contains a <simple comment>.

c) *P* contains a <dynamic parameter specification> whose data type is undefined as determined by the rules specified in Subclause 15.6, "<prepare statement>", in ISO/IEC 9075-5.

26) The data type of any <dynamic parameter specification> contained in *P* is determined by the rules specified in Subclause 15.6, "<prepare statement>", in ISO/IEC 9075-5.

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

28) *P* is prepared and the prepared statement is associated with FDW-execution.

29) If *P* is a <dynamic select statement> or a <dynamic single row select statement>, then:

    a) *P* becomes the select source associated with FDW-execution.

    b) A unique implementation-dependent name becomes the cursor name associated with FDW-execution.

30) The General Rules of Subclause 21.4, "Implicit DESCRIBE OUTPUT USING clause", are applied with *P* and *WRD* as *SOURCE* and *DESCRIPTOR*, respectively.

31) The General Rules of Subclause 21.3, "Implicit DESCRIBE INPUT USING clause", are applied with *P* and *WPD* as *SOURCE* and *DESCRIPTOR*, respectively.

Conformance Rules

None.

22.63 Iterate

Function

Retrieve the next row from a FDW-execution.

Definition

```
Iterate (
      ExecutionHandle     IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an opened FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error*.

3) Let *S* be the opened FDW-execution identified by ExecutionHandle.

4) Let *CR* be the open cursor associated with *S* and let *T* be the table associated with the open cursor.

5) Let *SRD* be the server row descriptor for *S* and let *N* be the value of the TOP_LEVEL_COUNT field of *SRD*.

6) For each item descriptor area in *SRD* whose LEVEL is 0 (zero) in the first *AD* item descriptor areas of *SRD*, and for all of their subordinate descriptor areas, refer to a <target specification> whose corresponding item descriptor area has a non-zero value of DATA_POINTER as a *bound target* and refer to the corresponding <select list> column as a *bound column*.

7) Let *IDA* be the item descriptor area of *SRD* corresponding to the *i*-th bound target and let *TT* be the value of the TYPE field of *IDA*.

8) If *TT* indicates DEFAULT, then:

   a) Let *WRD* be the wrapper row descriptor associated with *S*.

   b) Let *CT*, *P*, and *SC* be the values of the TYPE, PRECISION, and SCALE fields, respectively, for the item descriptor area of *WRD* corresponding to the *i*-th bound column.

   c) The data type, precision, and scale of the <target specification> described by *IDA* are effectively set to *CT*, *P*, and *SC*, respectively, for the purposes of this invocation of Iterate() only.

9) If *T* is empty, or if *CR* is positioned after the end of the result set, then:

   a) *CR* is positioned after the last row of *T*.

   b) No database values are assigned to bound targets.

   c) A completion condition is raised: *no data* and no further rules of this Subclause are applied.

10) Case:

    a) If the position of *CR* is before a row *NR*, then *CR* is positioned on row *NR*.

    b) If the position of *CR* is on a row *OR* other than the last row, then *CR* is positioned on the row immediately after *OR*. Let *NR* be the row immediately after *OR*.

11) *NR* becomes the current row of *CR*.

12) Case:

    a) If an exception condition is raised during derivation of any <derived column> associated with *NR*, then there is no fetched row associated with *S*, but *NR* remains the current row of *CR*.

    b) Otherwise:

        i) *NR* becomes the fetched row associated with *S*.

        ii) Let *SS* be the select source associated with *S*.

        iii) The General Rules of Subclause 21.7, "Implicit FETCH USING clause", are applied with *SS* and *S* as *SOURCE* and *ALLOCATED STATEMENT*, respectively.

        iv) If an exception condition is raised during the derivation of any target value, then the values of all the bound targets are implementation-dependent and *CR* remains positioned on the current row.

Conformance Rules

None.

22.64 Open

Function

Open an FDW-execution.

Definition

```
Open (
       ExecutionHandle     IN      INTEGER )
       RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) If *EH* identifies an opened FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error*.

4) Let *S* be the allocated FDW-execution identified by ExecutionHandle.

5) Case:

   a) If there is no prepared statement associated with *S*, then an exception condition is raised: *foreign-data wrapper-specific condition — function sequence error*.

   b) Otherwise, let *P* be the statement that was prepared.

6) *P* is executed.

7) Case:

   a) If *P* is a <dynamic select statement> or a <dynamic single row select statement>, then the General Rules of Subclause 21.2, "Implicit cursor", are applied to *P* and *S* as *SELECT SOURCE* and *ALLOCATED STATEMENT*, respectively.

   b) Otherwise:

      i) The General Rules of Subclause 21.5, "Implicit EXECUTE USING and OPEN USING clauses", are applied to 'EXECUTE', *P*, and *S*, as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

      ii) The results of the execution are the same as if the statement were contained in an <externally-invoked procedure> and executed; these are described in Subclause 13.3, "<externally-invoked procedure>", in ISO/IEC 9075-2.

      iii) If *P* is a <call statement>, then the General Rules of Subclause 21.6, "Implicit CALL USING clause", are applied to *P* and *S*, as *SOURCE* and *ALLOCATED STATEMENT*, respectively.

8) If *P* executed successfully, then any executed statement associated with *S* is destroyed and *P* becomes the executed statement associated with *S*.

9) *EH* is said to be an opened FDW-execution.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.65 ReOpen

Function

Reopen an FDW-execution.

Definition

```
ReOpen (
      ExecutionHandle     IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Let *AS* be the opened FDW-execution identified by *EH*.

4) Let *CN* be the name of the cursor associated with *AS*.

5) Cursor *CN* is re-opened in the following steps:

   a) Let *T* be the table specified by the select source associated with *AS*.

   b) Cursor *CN* is positioned before the first row of *T*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.66 RetrieveStatistics

Function

Retrieve implementation-defined statistics associated with a specified SQL-statement.

Definition

```
RetrieveStatistics (
      ExecutionHandle     IN      INTEGER,
      ReturnBuffer        OUT     CHARACTER VARYING(L),
      BufferLength        IN      INTEGER,
      ReturnedLength      OUT     INTEGER )
      RETURNS SMALLINT
```

where: *L* is determined by the value of ReturnedLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

General Rules

1) Let *EH* be the value of ExecutionHandle.

2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle.*

3) Case:

   a) If the foreign-data wrapper is able to report statistics associated with the foreign request associated with *EH*, then an XML document reporting those statistics is created.

   b) Otherwise, a completion condition is raised: *no data*.

4) Case:

   a) If the length in octets *LOXD* of the created XML document *XD* is greater than the value of BufferLength, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   b) Otherwise, the value of ReturnedLength is set to *LOXD* and the value of ReturnBuffer is set to *XD*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.67 SetDescriptor

Function

Set a field in an SQL/MED descriptor area.

Definition

```
SetDescriptor (
      DescriptorHandle    IN      INTEGER,
      RecordNumber        IN      SMALLINT,
      FieldIdentifier     IN      SMALLINT,
      Value               IN      ANY,
      BufferLength        IN      INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *D* be the allocated SQL/MED descriptor area identified by DescriptorHandle and let *N* be the value of the COUNT field of *D*.

2) The General Rules of Subclause 21.10, "Deferred parameter check", are applied to *D* as the *DESCRIPTOR AREA*.

3) Let *FI* be the value of FieldIdentifier.

4) If *FI* is not one of the code values in Table 33, "Codes used for SQL/MED descriptor fields", then an exception condition is raised: *foreign-data wrapper-specific condition — invalid descriptor field identifier*.

5) Let *DT* be the type of the descriptor *D*.

6) Let *MBS* be the value of the May Be Set column in the row of Table 36, "Ability to set SQL/MED descriptor fields", that contains *FI* and in the column that contains the descriptor type *DT*.

7) If *MBS* is 'No', then an exception condition is raised: *foreign-data wrapper-specific condition — invalid descriptor field identifier*.

8) Let *RN* be the value of RecordNumber.

9) Let *TYPE* be the value of the Type column in the row of Table 33, "Codes used for SQL/MED descriptor fields", that contains *FI*.

10) If TYPE is 'ITEM' and *RN* is less than 1 (one), then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

11) Let *IDA* be the item descriptor area of *D* specified by *RN*.

12) If an exception condition is raised in any of the following General Rules, then all fields of *IDA* for which specific values were provided in the invocation of SetDescField are set to implementation-dependent values and the value of COUNT for *D* is unchanged.

13) Information is set in *D*:

Case:

a) If *FI* indicates COUNT, then

   Case:

   i) If the memory requirements to manage the SQL/MED descriptor area cannot be satisfied, then an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error*.

   ii) Otherwise, the count of the number of item descriptor areas is set to the value of Value.

b) If *FI* indicates OCTET_LENGTH_POINTER, then the value of the OCTET_LENGTH_POINTER field of *IDA* is set to the address of Value.

c) If *FI* indicates DATA_POINTER, then the value of the DATA_POINTER field of *IDA* is set to the address of Value. If Value is a null pointer, then the address is set to 0 (zero).

d) If *FI* indicates INDICATOR_POINTER, then the value of the INDICATOR_POINTER field of *IDA* is set to the address of Value.

e) If *FI* indicates RETURNED_CARDINALITY_POINTER, then the value of the RETURNED_CARDINALITY_POINTER field of *IDA* is set to the address of Value.

f) If *FI* indicates CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, or CHARACTER_SET_NAME, then:

   i) Let *BL* be the value of BufferLength.

   ii) Case:

      1) If *BL* is not negative, then let *L* be *BL*.

      2) If *BL* indicates NULL TERMINATED, then let *L* be the number of octets of Value that precedes the implementation-defined null character that terminates a C character string.

      3) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

   iii) Case:

      1) If *L* is zero, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid string length or buffer length*.

      2) Otherwise, let *FV* be the first *L* octets of Value and let *TFV* be the value of

```
TRIM ( BOTH ' ' FROM FV )
```

   iv) Let *ML* be the maximum length in characters allowed for an <identifier> as specified in the Syntax Rules of Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2, and let *TFVL* be the length in characters of *TFV*.

v) Case:

1) If *TFVL* is greater than *ML*, then *FV* is set to the first *ML* characters of *TFV* and a completion condition is raised: *warning — string data, right truncation*.

2) Otherwise, *FV* is set to *TFV*.

vi) Case:

1) If *FI* indicates CHARACTER_SET_CATALOG and *FV* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid catalog name*.

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

3) If *FI* indicates CHARACTER_SET_NAME and *FV* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid character set name*.

vii) The value of the field of *IDA* identified by *FI* is set to the value of *FV*.

g) Otherwise, the value of the field of *IDA* identified by *FI* is set to the value of Value.

14) If *FI* indicates LEVEL, then:

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

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

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

ii) If Value is greater than *K*+1, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value*.

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

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

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

c) The value of LEVEL in *IDA* is set to Value.

15) If TYPE is 'ITEM' and *RN* is greater than *N*, then the COUNT field of *D* is set to *RN*.

16) If *FI* indicates TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, PARAMETER_MODE, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, PARAMETER_SPECIFIC_NAME, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, or SCOPE_NAME, then the DATA_POINTER field of *IDA* is set to 0 (zero).

17) If *FI* indicates DATA or if *FI* indicates DATA_POINTER, and Value is not a null pointer, and *IDA* is not consistent as specified in Subclause 21.1, "Description of MED item descriptor areas", then an exception condition is raised: *foreign-data wrapper-specific condition — inconsistent descriptor information*.

18) Let *V* be the value of Value.

19) If *FI* indicates TYPE, then:

    a) All the other fields of *IDA* are set to implementation-dependent values.

    b) Case:

        i) If *V* indicates CHARACTER, CHARACTER VARYING or CHARACTER LARGE OBJECT then the CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME fields of *IDA* are set to the values for the default character set name for the SQL-session and the LENGTH field of *IDA* is set to the maximum possible length in characters of the indicated data type.

        ii) If *V* indicates BIT or BIT VARYING, then the LENGTH field of *IDA* is set to the maximum possible length in bits of the indicated data type.

        iii) If V indicates BINARY LARGE OBJECT, then the LENGTH field of *IDA* is set to the maximum possible length in octets of the indicated data type.

        iv) If *V* indicates a <datetime type>, then the PRECISION field of *IDA* is set to 0 (zero).

        v) If *V* indicates INTERVAL, then the DATETIME_INTERVAL_PRECISION field of *IDA* is set to 2.

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

        vii) If *V* indicates SMALLINT or INTEGER, then the SCALE field of *IDA* is set to 0 (zero) and the PRECISION field of *IDA* is set to the implementation-defined value for the precision of the SMALLINT or INTEGER data types, respectively.

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

        ix) If *V* indicates REAL or DOUBLE PRECISION, then the PRECISION field of *IDA* is set to the implementation-defined value for the precision of the REAL or DOUBLE PRECISION data types, respectively.

        x) If *V* indicates an implementation-defined data type, then an implementation-defined set of fields of *IDA* are set to implementation-defined default values.

xi) Otherwise, an exception condition is raised: *foreign-data wrapper-specific condition — invalid data type.*

20) If *FI* indicates DATETIME_INTERVAL_CODE and the TYPE field of *IDA* indicates a <datetime type>, then:

a) All the fields of *IDA* other than DATETIME_INTERVAL_CODE and TYPE are set to implementation-dependent values.

b) Case:

i) If *V* indicates DATE, TIME, or TIME WITH TIME ZONE, then the PRECISION field of *IDA* is set to 0 (zero).

ii) If *V* indicates TIMESTAMP or TIMESTAMP WITH TIME ZONE, then the PRECISION field of *IDA* is set to 6.

21) If *FI* indicates DATETIME_INTERVAL_CODE and the TYPE field of *IDA* indicates INTERVAL, then the DATETIME_INTERVAL_PRECISION field of *IDA* is set to 2 and:

a) If *V* indicates DAY TO SECOND, HOUR TO SECOND, MINUTE TO SECOND, or SECOND, then the PRECISION field of *IDA* is set to 6.

b) Otherwise, the PRECISION field of *IDA* is set to 0 (zero).

Conformance Rules

The following restrictions apply for Core SQL:

None.

22.68 TransmitForeignRequest

Function

Supply an SQL-statement to be analyzed by the foreign-data wrapper and/or the foreign server.

Definition

```
TransmitForeignRequest (
      FSConnectionHandle  IN      INTEGER,
      RequestHandle       IN      INTEGER,
      ReplyHandle         OUT     INTEGER )
      RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle.

2) If *FSCH* does not identify an allocated FS-connection, then an exception condition is raised: *foreign-data wrapper-specific condition — invalid handle*.

3) Let *C* be the allocated FS-connection identified by *FSCH*.

4) The diagnostics area associated with *C* is emptied.

5) Let *RH* be the value of RequestHandle.

6) If the maximum number of FDW-replies that can be allocated at one time has already been reached, then ReplyHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — limit on number of handles exceeded*.

7) Case:

   a) If the memory requirements to manage an FDW-replies cannot be satisfied, then ReplyHandle is set to zero and an exception condition is raised: *foreign-data wrapper-specific condition — memory allocation error*.

   b) If the resources to manage an FDW-replies cannot be allocated for implementation-defined reasons, then ReplyHandle is set to zero and an implementation-defined exception condition is raised.

   c) Otherwise, the resources to manage an FDW-replies are allocated and are referred to as an allocated FDW-reply. The allocated FDW-reply is assigned a unique value *RHV* that is returned in ReplyHandle.

8) Case:

   a) If the foreign-data wrapper cannot create an FDW-reply that corresponds to the foreign request associated with *RH*, then an exception condition is raised: *foreign-data wrapper-specific condition – unable to create reply*.

   b) Otherwise, the FDW-reply corresponding to *RH* is created and the foreign request associated with *RH* is associated with *RHV*.

Conformance Rules

The following restrictions apply for Core SQL:

None.

23 Information Schema

23.1 COLUMN_OPTIONS view

Function

Identify the generic options specified for columns that are defined in this catalog.

Definition

```
CREATE VIEW COLUMN_OPTIONS AS
    SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           COLUMN_NAME, OPTION_NAME, OPTION_VALUE
    FROM DEFINITION_SCHEMA.COLUMN_OPTIONS
```

Conformance Rules

1) Without Feature M002, “Foreign-data wrapper”, conforming SQL language shall not reference the COLUMN_OPTIONS view.

23.2 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 IN
                   ( 'PUBLIC', CURRENT_USER )
                 OR
                   SCHEMA_OWNER IN
                   ( SELECT ROLE_NAME
                     FROM ENABLED_ROLES ) )
         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, D2.INTERVAL_TYPE)
            AS INTERVAL_TYPE,
      COALESCE (D1.INTERVAL_PRECISION, D2.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,
      COALESCE (D1.USER_DEFINED_TYPE_CATALOG, D2.USER_DEFINED_TYPE_CATALOG)
            AS USER_DEFINED_TYPE_CATALOG,
      COALESCE (D1.USER_DEFINED_TYPE_SCHEMA, D2.USER_DEFINED_TYPE_SCHEMA)
            AS USER_DEFINED_TYPE_SCHEMA,
      COALESCE (D1.USER_DEFINED_TYPE_NAME, D2.USER_DEFINED_TYPE_NAME)
```

```
          AS USER_DEFINED_TYPE_NAME,
      COALESCE (D1.SCOPE_CATALOG, D2.SCOPE_CATALOG)
          AS SCOPE_CATALOG,
      COALESCE (D1.SCOPE_SCHEMA, D2.SCOPE_SCHEMA)
          AS SCOPE_SCHEMA,
      COALESCE (D1.SCOPE_NAME, D2.SCOPE_NAME)
          AS SCOPE_NAME,
      COALESCE (D1.MAXIMUM_CARDINALITY, D2.MAXIMUM_CARDINALITY)
          AS MAXIMUM_CARDINALITY,
      COALESCE (D1.DTD_IDENTIFIER, D2.DTD_IDENTIFIER)
          AS DTD_IDENTIFIER,
      IS_SELF_REFERENCING,

      DATALINK_LINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION,
      DATALINK_WRITE_PERMISSION, DATALINK_RECOVERY, DATALINK_UNLINK
    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,
               'TABLE', C.DTD_IDENTIFIER )
           = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA, D1.OBJECT_NAME,
               D1.OBJECT_TYPE, D1.DTD_IDENTIFIER ) ) )
       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', C.DTD_IDENTIFIER )
          = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA, D2.OBJECT_NAME,
              D2.OBJECT_TYPE, D2.DTD_IDENTIFIER ) ) )
    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 ( SCHEMA_OWNER IN
                      ( 'PUBLIC', CURRENT_USER )
                  OR
                    SCHEMA_OWNER IN
                    ( SELECT ROLE_NAME
                      FROM ENABLED_ROLES ) ) )
      AND
          C.TABLE_CATALOG
        = ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE COLUMNS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature M001, "Datalinks", conforming SQL language shall not reference the columns DATALINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION, DATALINK_WRITE_PERMISSION, DATALINK_RECOVERY, or DATALINK_UNLINK.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the COLUMNS view.

23.3 FOREIGN_DATA_WRAPPER_OPTIONS view

Function

Identify the options specified for foreign-data wrappers that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_DATA_WRAPPER_OPTIONS AS
    SELECT FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM DEFINITION_SCHEMA.FOREIGN_DATA_WRAPPER_OPTIONS
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the FOREIGN_DATA_WRAPPER_OPTIONS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the FOREIGN_DATA_WRAPPER_OPTIONS view.

23.4 FOREIGN_DATA_WRAPPERS view

Function

Identify the foreign-data wrappers that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_DATA_WRAPPERS AS
    SELECT FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
           LIBRARY_NAME, LANGUAGE
    FROM DEFINITION_SCHEMA.FOREIGN_DATA_WRAPPERS;
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the FOREIGN_DATA_WRAPPERS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the FOREIGN_DATA_WRAPPERS view.

23.5 FOREIGN_SERVER_OPTIONS view

Function

Identify the options specified for foreign servers that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_SERVER_OPTIONS AS
    SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM DEFINITION_SCHEMA.FOREIGN_SERVER_OPTIONS
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the FOREIGN_SERVER_OPTIONS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the FOREIGN_SERVER_OPTIONS view.

23.6 FOREIGN_SERVERS view

Function

Identify the foreign servers defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_SERVERS AS
    SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
           FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
           FOREIGN_SERVER_TYPE, FOREIGN_SERVER_VERSION,
           AUTHORIZATION_IDENTIFIER
    FROM DEFINITION_SCHEMA.FOREIGN_SERVERS
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the FOREIGN_SERVERS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the FOREIGN_SERVERS view.

23.7 FOREIGN_TABLE_OPTIONS view

Function

Identify the options specified for foreign tables that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_TABLE_OPTIONS AS
    SELECT FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM DEFINITION_SCHEMA.FOREIGN_TABLE_OPTIONS
    WHERE ( FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME ) IN
          ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
            FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
            WHERE GRANTEE IN
                  ( 'PUBLIC', CURRENT_USER )
          UNION
          SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
          FROM DEFINITION_SCHEMA. COLUMN_PRIVILEGES
          WHERE GRANTEE IN
                ( 'PUBLIC', CURRENT_USER ) )
      AND
          TABLE_CATALOG
        = ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the FOREIGN_TABLE_OPTIONS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the FOREIGN_TABLE_OPTIONS view.

23.8 FOREIGN_TABLES view

Function

Identify the foreign tables that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_TABLES AS
    SELECT FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME,
           FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME
    FROM DEFINITION_SCHEMA.FOREIGN_TABLES
    WHERE ( FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME ) IN
          ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
            FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
            WHERE GRANTEE IN
                  ( 'PUBLIC', CURRENT_USER )
          UNION
            SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
            FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
            WHERE GRANTEE IN
                  ( 'PUBLIC', CURRENT_USER ) )
      AND
          TABLE_CATALOG
        = ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the FOREIGN_TABLES view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the FOREIGN_TABLES view.

23.9 USER_MAP_OPTIONS view

Function

Identify the options specified for user mappings that are defined in this catalog.

Definition

```
CREATE VIEW USER_MAP_OPTIONS AS
    SELECT AUTHORIZATION_IDENTIFIER,
           FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM DEFINITION_SCHEMA.USER_MAP_OPTIONS
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the USER_MAP_OPTIONS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the USER_MAP_OPTIONS view.

23.10 USER_MAPPINGS view

Function

Identify the user mappings that are defined in this catalog.

Definition

```
CREATE VIEW USER_MAPPINGS AS
    SELECT AUTHORIZATION_IDENTIFIER,
           FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME
    FROM DEFINITION_SCHEMA.USER_MAPPINGS
```

Conformance Rules

1) Without Feature M002, "Foreign-data wrapper", conforming SQL language shall not reference the USER_MAPPINGS view.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference the USER_MAPPINGS view.

23.11 Short name views

Function

Provide alternative views that use only identifiers that do not require Feature F391, "Long identifiers".

Definition

Replace COLUMNS_S with the following

```
CREATE VIEW COLUMNS_S
         ( TABLE_CATALOG,       TABLE_SCHEMA,        TABLE_NAME,
           COLUMN_NAME,         ORDINAL_POSITION,    COLUMN_DEFAULT,
           IS_NULLABLE,         DATA_TYPE,           CHAR_MAX_LENGTH,
           CHAR_OCTET_LENGTH,   NUMERIC_PRECISION,   NUMERIC_PREC_RADIX,
           NUMERIC_SCALE,       DATETIME_PRECISION,  INTERVAL_TYPE,
           INTERVAL_PRECISION,  CHAR_SET_CATALOG,    CHAR_SET_SCHEMA,
           CHARACTER_SET_NAME,  COLLATION_CATALOG,   COLLATION_SCHEMA,
           COLLATION_NAME,      DOMAIN_CATALOG,      DOMAIN_SCHEMA,
           DOMAIN_NAME,         UDT_CATALOG,         UDT_SCHEMA,
           UDT_NAME,            SCOPE_CATALOG,       SCOPE_SCHEMA,
           SCOPE_NAME,          MAX_CARDINALITY,     DTD_IDENTIFIER,

           IS_SELF_REF,         DL_LINK_CONTROL,     DL_INTEGRITY,
           DL_R_PERMISSION,     DL_W_PERMISSION,     DL_RECOVERY,
           DL_UNLINK ) AS

    SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           COLUMN_NAME, ORDINAL_POSITION, COLUMN_DEFAULT,
           IS_NULLABLE, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH,
           CHARACTER_OCTET_LENGTH, NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
           NUMERIC_SCALE, DATETIME_PRECISION, INTERVAL_TYPE,
           INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
           CHARACTER_SET_NAME, COLLATION_CATALOG, COLLATION_SCHEMA,
           COLLATION_NAME, DOMAIN_CATALOG, DOMAIN_SCHEMA,
           DOMAIN_NAME, UDT_CATALOG, UDT_SCHEMA,
           UDT_NAME, SCOPE_CATALOG, SCOPE_SCHEMA,
           SCOPE_NAME, MAXIMUM_CARDINALITY, DTD_IDENTIFIER,

           IS_SELF_REFERENCING,

           DATALINK_LINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION,
           DATALINK_WRITE_PERMISSION, DATALINK_RECOVERY, DATALINK_UNLINK
    FROM INFORMATION_SCHEMA.COLUMNS;

GRANT SELECT ON TABLE COLUMNS_S
    TO PUBLIC WITH GRANT OPTION;
```

Insert the following new short-name views

```
CREATE VIEW FD_WRAPPER_OPTS_S
         ( FDW_CATALOG,        FDW_NAME,           OPTION_NAME,
           OPTION_VALUE ) AS
    SELECT FOREIGN_DATA_WRAPPER_CATALOG FOREIGN_DATA_WRAPPER_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM INFORMATION_SCHEMA.FOREIGN_DATA_WRAPPER_OPTIONS

GRANT SELECT ON TABLE FD_WRAPPER_OPTS_S
    TO PUBLIC WITH GRANT OPTION;
```

23.11 Short name views

```
CREATE VIEW FD_WRAPPERS_S
         ( FDW_CATALOG,       FDW_NAME,           LIBRARY_NAME,
           LANGUAGE ) AS
    SELECT FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
           LIBRARY_NAME, LANGUAGE
    FROM INFORMATION_SCHEMA. FOREIGN_DATA_WRAPPERS;

GRANT SELECT ON TABLE FD_WRAPPERS_S
    TO PUBLIC WITH GRANT OPTION;

CREATE VIEW FGN_SERVER_OPTS_S
         ( FS_CATALOG,        FS_NAME,            OPTION_NAME,
           OPTION_VALUE ) AS
    SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM INFORMATION_SCHEMA.FOREIGN_SERVER_OPTIONS

GRANT SELECT ON TABLE FGN_SERVER_OPTS_S
    TO PUBLIC WITH GRANT OPTION;

CREATE VIEW FGN_TABLE_OPTS_S
         ( FT_CATALOG,        FT_SCHEMA,          FT_NAME,
           OPTION_NAME,       OPTION_VALUE ) AS
    SELECT FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM INFORMATION_SCHEMA.FOREIGN_TABLE_OPTIONS

GRANT SELECT ON TABLE FGN_TABLE_OPTS_S
    TO PUBLIC WITH GRANT OPTION;

CREATE VIEW FOREIGN_SERVERS
         ( FS_CATALOG,        FS_NAME,            FDW_CATALOG,
           FDW_NAME,          FS_TYPE             FS_VERSION,
           AUTH_ID,           PASSWORD ) AS
    SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME, FOREIGN_DATA_WRAPPER_CATALOG,
           FOREIGN_DATA_WRAPPER_NAME, FOREIGN_SERVER_TYPE, FOREIGN_SERVER_VERSION,
           AUTHORIZATION_IDENTIFIER, PASSWORD
    FROM INFORMATION_SCHEMA.FOREIGN_SERVERS

GRANT SELECT ON TABLE FGN_SERVERS_S
    TO PUBLIC WITH GRANT OPTION;

CREATE VIEW FOREIGN_TABLES
         ( FT_CATALOG,        FT_SCHEMA,          FT_NAME,
           FS_CATALOG,        FS_NAME ) AS
    SELECT FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME,
           FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME
    FROM INFORMATION_SCHEMA.FOREIGN_TABLES

GRANT SELECT ON TABLE FGN_TABLES_S
    TO PUBLIC WITH GRANT OPTION;

CREATE VIEW USER_MAP_OPTIONS
          ( AUTH_ID,          FS_CATALOG,         FS_NAME,
            OPTION_NAME       OPTION_VALUE ) AS
    SELECT AUTHORIZATION_IDENTIFIER, FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
           OPTION_NAME, OPTION_VALUE
    FROM INFORMATION_SCHEMA.USER_MAP_OPTIONS

GRANT SELECT ON TABLE USER_MAP_OPTS_S
    TO PUBLIC WITH GRANT OPTION;
```

```
CREATE VIEW USER_MAPPINGS
        ( AUTH_ID,           FS_CATALOG,         FS_NAME ) AS
    SELECT AUTHORIZATION_IDENTIFIER, FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME
    FROM INFORMATION_SCHEMA.USER_MAPPINGS

GRANT SELECT ON TABLE USER_MAPPINGS_S
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature M001, "Datalinks", conforming SQL language shall not reference the columns DATALINK_CONTROL, DL_INTEGRITY, DL_R_PERMISSION, DL_W_PERMISSION, DL_RECOVERY, or DL_UNLINK.

24 Definition Schema

24.1 COLUMN_OPTIONS base table

Function

The COLUMN_OPTIONS base table has one row for each option specified for each column.

Definition

```
CREATE TABLE COLUMN_OPTIONS (
    TABLE_CATALOG                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    TABLE_SCHEMA                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    TABLE_NAME                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    COLUMN_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_VALUE                    INFORMATION_SCHEMA.CHARACTER_DATA,

    CONSTRAINT COLUMN_OPTIONS_PRIMARY_KEY
      PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                    COLUMN_NAME, OPTION_NAME ),

    CONSTRAINT COLUMN_OPTIONS_FOREIGN_KEY_COLUMNS
      FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME )
        REFERENCES COLUMNS

)
```

Description

1) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalong name, unqualified schema name, qualified identifier, and identifier of the <column name> of the column whose option is being described.

2) The values of OPTION_NAME identify the option being described.

3) The values of OPTION_VALUE are the values specified for the option being described. The value of OPTION_VALUE is the null value if no value for the option being described was specified.

24.2 COLUMNS base table

Function

The COLUMNS table has one row for each column. It effectively contains a representation of the column descriptors.

Definition

```
CREATE TABLE COLUMNS (
    TABLE_CATALOG                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    TABLE_SCHEMA                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    TABLE_NAME                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    COLUMN_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    ORDINAL_POSITION                INFORMATION_SCHEMA.CARDINAL_NUMBER
      CONSTRAINT COLUMNS_ORDINAL_POSITION_NOT_NULL NOT NULL
      CONSTRAINT COLUMNS_ORDINAL_POSITION_GREATER_THAN_ZERO_CHECK
        CHECK ( ORDINAL_POSITION > 0 )
      CONSTRAINT COLUMNS_ORDINAL_POSITION_CONTIGUOUS_CHECK
        CHECK ( 0 = ALL ( SELECT MAX(ORDINAL_POSITION) - COUNT(*)
                          FROM COLUMNS
                          GROUP BY
                            TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) ),
    DOMAIN_CATALOG                  INFORMATION_SCHEMA.SQL_IDENTIFIER,
    DOMAIN_SCHEMA                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    DOMAIN_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    COLUMN_DEFAULT                  INFORMATION_SCHEMA.CHARACTER_DATA,
    IS_NULLABLE                     INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT COLUMNS_IS_NULLABLE_NOT_NULL NOT NULL
      CONSTRAINT COLUMNS_IS_NULLABLE_CHECK
        CHECK ( IS_NULLABLE IN ('YES', 'NO' ) ),
    IS_SELF_REFERENCING             INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT COLUMNS_IS_SELF_REFERENCING_NOT_NULL NOT NULL
      CONSTRAINT COLUMNS_IS_SELF_REFERENCING_CHECK
        CHECK ( IS_SELF_REFERENCING IN ('YES', 'NO' ) ),

    DATALINK_LINK_CONTROL           INFORMATION_SCHEMA.CHARACTER_DATA,
    DATALINK_INTEGRITY              INFORMATION_SCHEMA.CHARACTER_DATA,
    DATALINK_READ_PERMISSION        INFORMATION_SCHEMA.CHARACTER_DATA,
    DATALINK_WRITE_PERMISSION       INFORMATION_SCHEMA.CHARACTER_DATA,
    DATALINK_RECOVERY               INFORMATION_SCHEMA.CHARACTER_DATA,
    DATALINK_UNLINK                 INFORMATION_SCHEMA.CHARACTER_DATA,

    CONSTRAINT COLUMNS_PRIMARY_KEY
      PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

    CONSTRAINT COLUMNS_UNIQUE
      UNIQUE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, ORDINAL_POSITION ),

    CONSTRAINT COLUMNS_FOREIGN_KEY_TABLES
      FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
        REFERENCES TABLES,

    CONSTRAINT COLUMNS_CHECK_REFERENCES_DOMAIN
      CHECK ( DOMAIN_CATALOG
         NOT IN
              ( SELECT  CATALOG_NAME
                FROM SCHEMATA )
         OR
            ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
         IN
            ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
              FROM DOMAINS ) ),
```

```
    CONSTRAINT COLUMNS_DATALINK_LINKTYPE
      CHECK ( DATALINK_LINKTYPE IN ('URL') ),

    CONSTRAINT COLUMNS_DATALINK_LINK_CONTROL
      CHECK ( DATALINK_LINK_CONTROL IN ('YES, 'NO') ),

    CONSTRAINT COLUMNS_DATALINK_INTEGRITY
      CHECK ( DATALINK_INTEGRITY IN ('ALL', 'SELECTIVE', 'NONE') ),

    CONSTRAINT COLUMNS_DATALINK_READ_PERMISSION
      CHECK ( DATALINK_READ_PERMISSION IN ('FS', 'DB') ),

    CONSTRAINT COLUMNS_DATALINK_WRITE_PERMISSION
      CHECK ( DATALINK_WRITE_PERMISSION IN ('FS', 'BLOCKED') ),

    CONSTRAINT COLUMNS_DATALINK_RECOVERY
      CHECK ( DATALINK_RECOVERY IN ('YES', 'NO') ),

    CONSTRAINT COLUMNS_DATALINK_UNLINK
      CHECK ( DATALINK_UNLINK IN ('DELETE', 'RESTORE', 'NONE') ),

    CONSTRAINT COLUMNS_CHECK_DATA_TYPE
      CHECK ( DOMAIN_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
                  IS NOT NULL
              AND
                ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) NOT IN
                ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, COLUMN_NAME
                  FROM DATA_TYPE_DESCRIPTOR )
             OR
                ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
                  IS NULL
              AND
                ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
                ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, COLUMN_NAME
                  FROM DATA_TYPE_DESCRIPTOR )
              )
            )
)
```

Description

1) Insert this Description If the data type of the column being defined is not a datalink data type, then the values of DATALINK_LINKTYPE, DATALINK_LINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION, DATALINK_WRITE_PERMISSION, DATALINK_RECOVER, and DATALINK_UNLINK are the null value; otherwise, the values of DATALINK_LINKTYPE, DATALINK_LINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION, DATALINK_WRITE_PERMISSION, DATALINK_RECOVER, and DATALINK_UNLINK are the link type, link control, integrity control option, read permission option, write permission option, recovery option, and unlink option of the datalink data type used by the column being described.

2) Insert this Description The values of DATALINK_LINK_CONTROL have the following meanings:

YES	The datalink value in the column is under file link control.
NO	The datalink value in the column is not under file link control.

24.2 COLUMNS base table

3) Insert this Description The values of DATALINK_INTEGRITY have the following meanings:

ALL	The external file corresponding to the datalink value in the column is under the control of the SQL-implementation.
SELECTIVE	The external file corresponding to the datalink value in the column is under the control of the SQL-implementation in an implementation-dependent manner.
NONE	The external file corresponding to the datalink value in the column is not under the control of the SQL-implementation.

4) Insert this Description The values of DATALINK_READ_PERMISSION have the following meanings:

FS	The external file corresponding to the datalink value in the column is under the operating system's file programming permissions for read access.
DB	The external file corresponding to the datalink value in the column is under the SQL-implementation's control for read access.

5) Insert this Description The values of DATALINK_WRITE_PERMISSION have the following meanings:

FS	The external file corresponding to the datalink value in the column is under the operating system's file programming permissions for write access.
BLOCKED	Write access to the external file corresponding to the datalink value in the column is blocked.

6) Insert this Description The values of DATALINK_RECOVERY have the following meanings:

YES	The coordinated recovery of SQL-implementation data and external files is supported.
NO	The coordinated recovery of SQL-implementation data and external files is not supported.

7) Insert this Description The values of DATALINK_UNLINK have the following meanings:

DELETE	On unlink, the external file corresponding to the datalink value in the column is deleted.
RESTORE	On unlink, the file attributes of the external file corresponding to the datalink value in the column are restored to those at the time when the file was linked.
NONE	The external file corresponding to the datalink value in the column is not under SQL-implementation control.

24.3 DATA_TYPE_DESCRIPTOR base table

Function

The DATA_TYPE_DESCRIPTOR table has one row for each domain, one row for each column (in each table) and for each attribute (in each structured type) that is defined as having a data type rather than a domain, one row for each distinct type, one row for the result type of each SQL parameter of each SQL-invoked routine, one row for the result type of each method specification, one row for each parameter of each method specification, and one row for each structured type whose associated reference type has a user-defined representation. It effectively contains a representation of the data type descriptors.

Definition

Replace constraint DATA_TYPE_DESCRIPTOR_DATA_TYPE_CHECK_COMBINATIONS with

```
CONSTRAINT DATA_TYPE_DESCRIPTOR_DATA_TYPE_CHECK_COMBINATIONS
  CHECK ( ( DATA_TYPE IN
              ( 'CHARACTER', 'CHARACTER VARYING', 'CHARACTER LARGE OBJECT' )
            AND
              ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NOT NULL
            AND
              ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
                NUMERIC_SCALE, DATETIME_PRECISION,
                USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                USER_DEFINED_TYPE_NAME ) IS NULL
            AND
              ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
            AND
              ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
            AND
              MAXIMUM_CARDINALITY IS NULL )
          OR
            ( DATA_TYPE IN
              ( 'BIT', 'BIT VARYING' )
            AND
              ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH ) IS NOT NULL
            AND
              ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
            AND
              ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
                NUMERIC_SCALE, DATETIME_PRECISION,
                USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                USER_DEFINED_TYPE_NAME ) IS NULL
            AND
              ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
            AND
              ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
            AND
              MAXIMUM_CARDINALITY IS NULL )
          OR
            ( DATA_TYPE IN
              ( 'BINARY LARGE OBJECT' )
            AND
              ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH )
                  IS NOT NULL
            AND
              ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
                NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
                NUMERIC_SCALE, DATETIME_PRECISION,
                USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
```

```
        USER_DEFINED_TYPE_NAME ) IS NULL
    AND
      ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
    AND
      ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE IN
      ( 'INTEGER', 'SMALLINT' )
    AND
      ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
        COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
    AND
      NUMERIC_PRECISION_RADIX IN
      ( 2, 10 )
    AND
      NUMERIC_PRECISION IS NOT NULL
    AND
      NUMERIC_SCALE = 0
    AND
      DATETIME_PRECISION IS NULL
    AND
      ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
    AND
      ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE IN
      ( 'NUMERIC', 'DECIMAL' )
    AND
      ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
        COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
    AND
      NUMERIC_PRECISION_RADIX = 10
    AND
      ( NUMERIC_PRECISION, NUMERIC_SCALE ) IS NOT NULL
    AND
      DATETIME_PRECISION IS NULL
    AND
      ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
    AND
      ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE IN
      ( 'REAL', 'DOUBLE PRECISION', 'FLOAT' )
    AND
      ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
        COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
    AND
      NUMERIC_PRECISION IS NOT NULL
    AND
      NUMERIC_PRECISION_RADIX = 2
    AND
      NUMERIC_SCALE IS NULL
    AND
      DATETIME_PRECISION IS NULL
    AND
      ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME ) IS NULL
    AND
```

```
    ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
  AND
    ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
  AND
    MAXIMUM_CARDINALITY IS NULL )
OR
  ( DATA_TYPE IN
    ( 'DATE', 'TIME', 'TIMESTAMP',
      'TIME WITH TIME ZONE', 'TIMESTAMP WITH TIME ZONE' )
  AND
    ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
      COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
  AND
    ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
  AND
    NUMERIC_SCALE IS NULL
  AND
    DATETIME_PRECISION IS NOT NULL
  AND
    ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
      USER_DEFINED_TYPE_NAME ) IS NULL
  AND
    ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
  AND
    ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
  AND
    MAXIMUM_CARDINALITY IS NULL )
OR
  ( DATA_TYPE = 'INTERVAL'
  AND
    ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
      COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
  AND
    ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
  AND
    NUMERIC_SCALE IS NULL
  AND
    DATETIME_PRECISION IS NOT NULL
  AND
    ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
      USER_DEFINED_TYPE_NAME ) IS NULL
  AND
    INTERVAL_TYPE IN
    ( 'YEAR', 'MONTH', 'DAY', 'HOUR', 'MINUTE', 'SECOND',
      'YEAR TO MONTH', 'DAY TO HOUR', 'DAY TO MINUTE', 'DAY TO SECOND', 'HOUR TO MINUTE',
      'HOUR TO SECOND', 'MINUTE TO SECOND' )
  AND
    INTERVAL_PRECISION IS NOT NULL
  AND
    ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
  AND
    MAXIMUM_CARDINALITY IS NULL )
OR
  ( DATA_TYPE IN ( 'BOOLEAN', 'DATALINK' )
  AND
    ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
      COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IS NULL
  AND
    ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NULL
  AND
    NUMERIC_SCALE IS NULL
  AND
    DATETIME_PRECISION IS NULL
  AND
```

```
      ( INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
    AND
      ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME ) IS NULL
    AND
      ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE = 'USER-DEFINED'
    AND
      ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
        NUMERIC_SCALE, DATETIME_PRECISION, CHARACTER_OCTET_LENGTH,
        CHARACTER_MAXIMUM_LENGTH, INTERVAL_TYPE, INTERVAL_PRECISION,
        SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME ) IS NOT NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE = 'REF'
    AND
      ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH ) IS NOT NULL
    AND
      ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
        NUMERIC_SCALE, DATETIME_PRECISION,
        INTERVAL_TYPE, INTERVAL_PRECISION ) IS NULL
    AND
      ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME ) IS NOT NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE = 'ARRAY'
    AND
      ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
        NUMERIC_SCALE, DATETIME_PRECISION, CHARACTER_OCTET_LENGTH,
        CHARACTER_MAXIMUM_LENGTH, INTERVAL_TYPE, INTERVAL_PRECISION )
          IS NULL
    AND
      ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME ) IS NULL
    AND
      ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      MAXIMUM_CARDINALITY IS NOT NULL )
  OR
    ( DATA_TYPE = 'ROW'
    AND
      ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
        NUMERIC_SCALE, DATETIME_PRECISION, CHARACTER_OCTET_LENGTH,
        CHARACTER_MAXIMUM_LENGTH, INTERVAL_TYPE, INTERVAL_PRECISION )
          IS NULL
    AND
      ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME ) IS NULL
    AND
      ( SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME ) IS NULL
    AND
      MAXIMUM_CARDINALITY IS NULL )
  OR
    ( DATA_TYPE NOT IN
      ( 'CHARACTER', 'CHARACTER VARYING', 'CHARACTER LARGE OBJECT',
```

```
'BINARY LARGE OBJECT',
'BIT', 'BIT VARYING',
'INTEGER', 'SMALLINT', 'NUMERIC', 'DECIMAL',
'REAL', 'DOUBLE PRECISION', 'FLOAT',
'DATE', 'TIME', 'TIMESTAMP',
'INTERVAL', 'BOOLEAN', 'USER-DEFINED',
'REF', 'ARRAY', 'ROW' ) ),
```

Description

1) Insert this Description If DATA_TYPE is 'DATALINK', then the data type being described is the datalink type.

24.4 FOREIGN_DATA_WRAPPER_OPTIONS base table

Function

The FOREIGN_DATA_WRAPPER_OPTIONS base table has one row for each option specified for each foreign-data wrapper.

Definition

```
CREATE TABLE FOREIGN_DATA_WRAPPER_OPTIONS (
    FOREIGN_DATA_WRAPPER_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_DATA_WRAPPER_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_NAME                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_VALUE                       INFORMATION_SCHEMA.SQL_CHARACTER_DATA,

    CONSTRAINT FOREIGN_DATA_WRAPPER_OPTIONS_PRIMARY_KEY
      PRIMARY KEY ( FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
                    OPTION_NAME ),

    CONSTRAINT FOREIGN_DATA_WRAPPER_OPTIONS_FOREIGN_KEY_FOREIGN_DATA_WRAPPER
      FOREIGN KEY ( FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME )
        REFERENCES FOREIGN_DATA_WRAPPERS
)
```

Description

1) The values of FOREIGN_DATA_WRAPPER_CATALOG and FOREIGN_DATA_WRAPPER_NAME are the catalog name and qualified identifier, respectively, of the foreign-data wrapper for which the option being described is specified.

2) The value for OPTION_NAME identifies the option being described.

3) The value for OPTION_VALUE is the value specified for the option being described. The value of OPTION_VALUE is the null value if no value for the option being described was specified.

24.5 FOREIGN_DATA_WRAPPERS base table

Function

The FOREIGN_DATA_WRAPPERS base table has one row for each foreign-data wrapper.

Definition

```
CREATE TABLE FOREIGN_DATA_WRAPPERS (
    FOREIGN_DATA_WRAPPER_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_DATA_WRAPPER_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
    LIBRARY_NAME                        INFORMATION_SCHEMA.SQL_CHARACTER_DATA,
    LANGUAGE                            INFORMATION_SCHEMA.SQL_CHARACTER_DATA,

    CONSTRAINT FOREIGN_DATA_WRAPPERS_PRIMARY_KEY
      PRIMARY KEY ( FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME ),

    CONSTRAINT FOREIGN_DATA_WRAPPERS_LANGUAGE_CHECK
      CHECK ( LANGUAGE IN
        ( 'ADA', 'C', 'COBOL', 'FORTRAN',
          'MUMPS', 'PASCAL', 'PLI' ) )
)
```

Description

1) The values of FOREIGN_DATA_WRAPPER_CATALOG and FOREIGN_DATA_WRAPPER_NAME are the catalog name and qualified identifier, respectively, of the foreign-data wrapper being described.

2) The value of LIBRARY_NAME is the name of the library that contains the entry points of the routines of the foreign-data wrapper being described.

3) The value of the LANGUAGE is the language of the routines of the foreign-data wrapper being described.

24.6 FOREIGN_SERVER_OPTIONS base table

Function

The FOREIGN_SERVER_OPTIONS base table has one row for each option specified for each foreign server.

Definition

```
CREATE TABLE FOREIGN_SERVER_OPTIONS (
    FOREIGN_SERVER_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_NAME                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_VALUE                     INFORMATION_SCHEMA.SQL_CHARACTER_DATA,

    CONSTRAINT FOREIGN_SERVERS_PRIMARY_KEY
      PRIMARY KEY ( FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME, OPTION NAME ),

    CONSTRAINT FOREIGN_SERVER_OPTIONS_FOREIGN_KEY_FOREIGN_SERVER
      FOREIGN KEY ( FOREIGN_SERVER _CATALOG, FOREIGN_SERVER_NAME )
        REFERENCES FOREIGN_SERVERS
)
```

Description

1) The values of FOREIGN_SERVER_CATALOG and FOREIGN_SERVER_NAME are the catalog name and qualified identifier, respectively, of the foreign server for which the option being described is specified.

2) The value for OPTION_NAME identifies the option being described.

3) The value for OPTION_VALUE is the value specified for the option being described. The value of OPTION_VALUE is the null value if no value for the option being described was specified.

24.7 FOREIGN_SERVERS base table

Function

The FOREIGN_SERVERS base table has one row for each foreign server.

Definition

```
CREATE TABLE FOREIGN_SERVERS (
    FOREIGN_SERVER_CATALOG             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_NAME                INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_DATA_WRAPPER_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER
      CONSTRAINT FOREIGN_SERVER_FOREIGN_DATA_WRAPPER_CATALOG_NOT_NULL NOT NULL,
    FOREIGN_DATA_WRAPPER_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER
      CONSTRAINT FOREIGN_SERVER_FOREIGN_DATA_WRAPPER_NAME_NOT_NULL NOT NULL,
    FOREIGN_SERVER_TYPE                INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT FOREIGN_SERVER_TYPE_NOT_NULL NOT NULL,
    FOREIGN_SERVER_VERSION             INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT FOREIGN_SERVER_VERSION_NOT_NULL NOT NULL,
    AUTHORIZATION_IDENTIFIER           INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PASSWORD                           INFORMATION_SCHEMA.CHARACTER_DATA,

    CONSTRAINT FOREIGN_SERVERS_PRIMARY_KEY
      PRIMARY KEY ( FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME ),

    CONSTRAINT FOREIGN_SERVERS_FOREIGN_KEY_FOREIGN_DATA_WRAPPER
      FOREIGN KEY ( FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME )
        REFERENCES FOREIGN_DATA_WRAPPERS
)
```

Description

1) The values of FOREIGN_SERVER_CATALOG and FOREIGN_SERVER_NAME are the catalog name and qualified identifier, respectively, of the foreign server being described.

2) The values of FOREIGN_DATA_WRAPPER_CATALOG and FOREIGN_DATA_WRAPPER_NAME are the catalog name and qualified identifier, respectively, of the foreign-data wrapper used when the foreign server being described is accessed.

3) The value of FOREIGN_SERVER_TYPE identifies the type of the foreign server being described.

4) The value of FOREIGN_SERVER_VERSION identifies the version of the type of the foreign server being described.

5) If the value of AUTHORIZATION_IDENTIFIER is not the null value, then it is the authorization identifier to be used when a connection is established to the foreign server being described.

6) If the value of PASSWORD is not the null value, then it is the password to be used when a connection is established to the foreign server being described.

24.8 FOREIGN_TABLE_OPTIONS base table

Function

The FOREIGN_TABLE_OPTIONS base table has one row for each option specified for each foreign table.

Definition

```
CREATE TABLE FOREIGN_TABLE_OPTIONS (
    FOREIGN_TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_NAME                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_VALUE                   INFORMATION_SCHEMA.SQL_CHARACTER_DATA,

    CONSTRAINT FOREIGN_TABLE_OPTIONS_PRIMARY_KEY
      PRIMARY KEY ( FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA,
                    FOREIGN_TABLE_NAME, OPTION NAME ),

    CONSTRAINT FOREIGN_TABLE_OPTIONS_FOREIGN_KEY_FOREIGN_TABLE
      FOREIGN KEY (FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME )
        REFERENCES FOREIGN_TABLES
)
```

Description

1) The values of FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA and FOREIGN_TABLE_NAME are the catalog name and qualified identifier, respectively, of the foreign table for which the option being described is specified.

2) The value for OPTION_NAME identifies the option being described.

3) The value for OPTION_VALUE is the value specified for the option being described. The value of OPTION_VALUE is the null value if no value for the option being described was specified.

24.9 FOREIGN_TABLES base table

Function

The FOREIGN_TABLES base table has one row for each foreign table.

Definition

```
CREATE TABLE FOREIGN_TABLES (
    FOREIGN_TABLE_CATALOG             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_TABLE_SCHEMA              INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_TABLE_NAME                INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT FOREIGN_TABLES_PRIMARY_KEY
      PRIMARY KEY ( FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA,
                    FOREIGN_TABLE_NAME ),

    CONSTRAINT FOREIGN_TABLES_FOREIGN_KEY_FOREIGN_SERVER
      FOREIGN KEY ( FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME )
        REFERENCES FOREIGN_SERVERS
)
```

Description

1) The values of FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA and FOREIGN_TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the foreign table being described.

2) The values of FOREIGN_SERVER_CATALOG and FOREIGN_SERVER_NAME are the catalog name and qualified identifier, respectively, of the foreign server that is the source of the foreign table being described.

24.10 SQL_SIZING base table

Function

The SQL_SIZING base table has one row for each sizing item defined by ISO/IEC 9075.

Definition

No additional Definition items

Description

No additional Descriptions.

Table population

Add the following item to the list of INSERT values:

1)

```
( 20004, 'MAXIMUM DATALINK LENGTH',
      'Length in octets' ),
```

24.11 TABLES base table

Function

The TABLES base table contains one row for each table, including views and foreign tables. It effectively contains a representation of the table descriptors.

Definition

Augment the column constraint TABLE_TYPE_CHECKin ISO/IEC 9075-5 Add , 'FOREIGN'to the <in value list> of valid REFERENCE_GENERATION values.

Description

1) Augment Description 3)

FOREIGN	The table being described is a foreign table.

24.12 USAGE_PRIVILEGES base table

Function

The USAGE_PRIVILEGES table has one row for each usage privilege descriptor. It effectively contains a representation of the usage privilege descriptors.

Definition

No additional Definition items.

Description

1) Replace D3) Case:

   a) If the object to which the privileges apply is a foreign-data wrapper or a foreign server, then the values of OBJECT_CATALOG and OBJECT_NAME are the catalog name, and qualified identifier, respectively, of the object to which the privilege applies and OBJECT_SCHEMA is the empty string.

   b) Otherwise, the values of OBJECT_CATALOG, OBJECT_SCHEMA, and OBJECT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the object to which the privilege applies.

24.13 USER_MAPPING_OPTIONS base table

Function

The USER_MAPPING_OPTIONS base table has one row for each option specified for each user mapping.

Definition

```
CREATE TABLE USER_MAPPING_OPTIONS (
    AUTHORIZATION_IDENTIFIER      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_NAME                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    OPTION_VALUE                  INFORMATION_SCHEMA.SQL_CHARACTER_DATA,

    CONSTRAINT USER_MAPPING_OPTIONS_PRIMARY_KEY
      PRIMARY KEY ( OPTION_NAME, AUTHORIZATION_IDENTIFIER,
                    FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME ),

    CONSTRAINT USER_MAPPING_OPTIONS_FOREIGN_KEY_MAPPING
      FOREIGN KEY ( AUTHORIZATION_IDENTIFIER, FOREIGN_SERVER_CATALOG,
                    FOREIGN_SERVER_NAME )
        REFERENCES USER_MAPPINGS
)
```

Description

1) The values of AUTHORIZATION_IDENTIFIER, FOREIGN_SERVER_CATALOG and FOREIGN_SERVER_NAME are the authorization identifier, the catalog name and qualified identifier, respectively, of the user mapping for for which the option being described is specified.

2) The value for OPTION_NAME identifies the option being described.

3) The value for OPTION_VALUE is the value specified for the option being described. The value of OPTION_VALUE is the null value if no value for the option being described was specified.

24.14 USER_MAPPINGS base table

Function

The USER_MAPPINGS base table has one row for each user mapping.

Definition

```
CREATE TABLE USER_MAPPINGS (
    AUTHORIZATION_IDENTIFIER       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER,
    FOREIGN_SERVER_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT USER_MAPPINGS_PRIMARY_KEY
      PRIMARY KEY ( AUTHORIZATION_IDENTIFIER, FOREIGN_SERVER_CATALOG,
                    FOREIGN_SERVER_NAME ),

    CONSTRAINT USER_MAPPINGS_FOREIGN_KEY_FOREIGN_SERVER
      FOREIGN KEY ( FOREIGN_SERVER _CATALOG, FOREIGN_SERVER_NAME )
        REFERENCES FOREIGN_SERVERS
```

Description

1) The value of AUTHORIZATION_IDENTIFIER identifies the authorization identifier whose user mapping for the foreign server identified by FOREIGN_SERVER_CATALOG and FOREIGN_ SERVE_NAME is being described.

2) The values of FOREIGN_SERVER_CATALOG and FOREIGN_SERVER_NAME are the catalog name and qualified identifier, respectively, of the foreign server for which the user mapping is being described.

25 Status codes

25.1 SQLSTATE

Table 38—SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
	All alternatives from ISO/IEC 9075-2			
	All alternatives from ISO/IEC 9075-5			
X	data exception	22	(*no subclass*)	000
			invalid Link Type	016
			invalid data specified for datalink	017
			null argument passed to datalink constructor	01A
			datalink value exceeds maximum length	01D

26 Conformance

26.1 Conformance to SQL/MED

This part of ISO/IEC 9075 specifies conforming SQL/MED language, SQL/MED routines and conforming SQL/MED implementations.

Conforming SQL/MED language shall abide by the Format, associated Syntax Rules and Access Rules, Definitions, and Descriptions, and shall abide by the restrictions imposed by all Conformance Rules.

A conforming SQL/MED implementation shall support at least one of Feature M003, "Datalinks via SQL/CLI", Feature M004, "Datalinks via SQL language", and Feature M002, "Foreign-data wrappers".

A conforming SQL/MED-implementation shall process conforming SQL/MED language according to the associated General Rules, Definitions, and Descriptions and shall invoke SQL/CLI routines and SQL/MED foreign data wrapper routines specified in this part of ISO/IEC 9075. Such routine invocations shall be constructed according to the BNF Format and associated Syntax Rules, Access Rules, and Definitions specified for <CLI routine>s in Clause 19, "Call-Level Interface specifications", Clause 20, "SQL/CLI routines", and Clause 22, "Foreign-data wrapper interface routines", in this part of ISO/IEC 9075.

A feature *FEAT1* may imply another feature *FEAT2*. An SQL-implementation that claims to support *FEAT1* shall also support each feature *FEAT2* implied by *FEAT1*. Conversely, an application need only designate that it requires *FEAT1*, and may assume that this includes each feature *FEAT2* implied by *FEAT1*. The list of features that are implied by other features is shown in Table 39, "Implied feature relationships". Note that some features imply multiple other features.

Table 39—Implied feature relationships

Feature ID	Feature Description	Implied Feature ID	Implied Feature Description
M003	Datalinks via SQL/CLI	M001	Datalinks
M004	Datalinks via Embedded SQL	M001	Datalinks

26.2 Claims of conformance

Insert this paragraph Claims of conformance to this part of ISO/IEC 9075 shall state:

1) Insert after list element 2) in ISO/IEC 9075-1 Whether or not Feature M001, "Datalinks", is supported

2) Insert after list element 2) in ISO/IEC 9075-1 Whether or not Feature M002, "Foreign-data wrappers", is supported and, if so, for which of the following programming languages:

   a) Ada

b) C

c) COBOL

d) Fortran

e) MUMPS

f) Pascal

g) PL/I

3) Insert after list element 2) in ISO/IEC 9075-1 Whether Feature M003, "Datalinks via SQL/CLI", is supported and, if so, for which of the following programming languages:

a) Ada

b) C

c) COBOL

d) Fortran

e) MUMPS

f) Pascal

g) PL/I

4) Insert after list element 2) in ISO/IEC 9075-1 Whether Feature M004, "Datalinks via SQL language", is supported and, if so, for which of the following programming languages:

a) Ada

b) C

c) COBOL

d) Fortran

e) MUMPS

f) Pascal

g) PL/I

5) Insert after list element 2) in ISO/IEC 9075-1 The definitions for all elements and actions that are specified in this part of ISO/IEC 9075 as implementation-defined.

26.3 Extensions and options

New paragraph A conforming implementation may provide support for additional implementation-defined routines or for implementation-defined argument values for <foreign data wrapper routine>s.

New paragraph An implementation remains conforming even if it provides user options to process conforming <foreign data wrapper routine> invocations in a nonconforming manner.

Annex A
(informative)

SQL Conformance Summary

The contents of this Annex summarizes all Conformance Rules, ordered by Feature ID and by Subclause.

****Editor's Note****
The contents of this Annex will be automatically generated before FDIS ballot initiation, but will remain empty until that time.

Annex B
(informative)

Implementation-defined elements

This Annex references those features that are identified in the body of this part of ISO/IEC 9075 as implementation-defined.

The term *implementation-defined* is used to identify characteristics that may differ between implementations, but that shall be defined for each particular implementation.

- 1 list element deleted.

1) Insert this list element Subclause 4.14.1, "Handles":

   a) The validity of a handle in a compilation unit other than the one in which the identified resource was allocated is implementation-defined.

2) Insert this list element Subclause 4.14.4, "Diagnostics areas in SQL/MED":

   a) If the routine's return code indicates **No data found**, then no status record is generated corresponding to SQLSTATE value '02000' but there may be status records generated corresponding to SQLSTATE value '02*nnn*', where '*nnn*' is an implementation-defined subclass value.

3) Insert this list element Subclause 4.5, "Generic options":

   a) Both the option name and the option value of a generic option are implementation-defined.

4) Insert this list element Subclause 4.7, "Datalinks":

   a) The time at which a valid access token ceases to be valid is implementation-defined.

   b) The implementation-defined *maximum datalink length* determines the amount of space, in octets, that is allocated for:

      — A host variable of data type DATALINK.

      — An argument of declared type DATALINK to an invocation of an external routine.

      — The value returned by an invocation of an external function whose result type is DATALINK.

      The maximum datalink length constrains the values of expressions whose declared type is DATALINK such that every such value can be assigned to a host variable, substituted for a parameter to an external routine, or returned by an invocation of an external function.

5) Insert this list element Subclause 4.2, "Foreign servers":

a) The possible values of server type and server version, and their meanings, are implementation-defined.

- 1 list element deleted.

- 3 list elements deleted.

6) Insert this list element Subclause 4.14.5, "Null terminated strings":

a) The null character that terminates C character strings is implementation-defined.

7) Insert this list element Table 4, "Fields used in SQL/MED diagnostics areas":

a) The maximum lengths of SQL/MED diagnostics area fields whose data types are CHARACTER VARYING are implementation-defined.

b) SQL/MED supports implementation-defined header fields in SQL/MED diagnostics areas.

8) Insert this list element Table 32, "Codes used for SQL/MED diagnostic fields":

a) SQL/MED supports implementation-defined diagnostics header fields and implementation-defined diagnostics status fields.

- 1 list element deleted.

9) Augment list element 19) Subclause 6.3, "<string value function>":

a) Insert this list element

b) If <datalink comment expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

c) Insert this list element If <link type expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

d) Insert this list element If <url complete expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

e) Insert this list element If <url path expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

f) Insert this list element If <url path only expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

g) Insert this list element If <url scheme expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

h) Insert this list element If <url server expression> is specified, then the data type of the result is a variable-length character string with an implementation-defined maximal length.

10) Insert this list element Subclause 11.2, "<generic options>":

a) The permissible <option name>s are implementation-defined.

b) The permissible <option value>s are implementation-defined.

11) Insert this list element Subclause 12.6, "<foreign table definition>":

a) If <basic column definition list> is specified, then the nullability characteristic and <default option> of each column specified by <basic column definition> is implementation-defined.

b) If <basic column definition list> is not specified, then column descriptors included in a foreign table descriptor are implementation-defined.

12) Insert this list element Subclause 12.7, "<alter foreign table statement>":

a) If <alter generic options> is specified, then any effect on the foreign table descriptor, apart from its generic options descriptor, is implementation-defined.

13) Insert this list element Subclause 12.8, "<add basic column definition>":

a) The nullability characteristic and <default option> included in the column descriptor specified by <basic column definition> is implementation-defined.

14) Augment list element 16), in ISO/IEC 9075-5) Subclause 15.8, "<describe statement>": If TYPE indicates DATALINK, then LENGTH is set to the length of maximum length in characters of the character string; OCTET_LENGTH is set to the maximum possible length in octets of the character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; and the <collation name> of the character string's collation. If the subject <language clause> specifies C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string.

15) Insert this list element Subclause 13.1, "<foreign server definition>":

a) The permissible Format and values for <server type>, <server version> and <password> are implementation-defined.

b) The privileges necessary to execute <foreign server definition> are implementation-defined.

16) Insert this list element Subclause 13.4, "<foreign-data wrapper definition>":

a) The privileges necessary to execute <foreign-data wrapper definition> are implementation-defined.

17) Insert this list element Subclause 21.1, "Description of MED item descriptor areas":

a) The null character that terminates C character strings is implementation-defined.

b) Let IDA be an item descriptor area in a wrapper parameter descriptor. One condition that allows *IDA* to be valid is if TYPE indicates an implementation-defined data type.

c) One condition that allows an SQL/MED item descriptor area in an SQL/MED descriptor area that is not a wrapper row descriptor to be consistent is if TYPE indicates an implementation-defined data type.

d) Let *IDA* be an item descriptor area in a server parameter descriptor. One condition that allows *IDA* to be valid is if TYPE indicates an implementation-defined data type.

e) One condition that allows an SQL/MED item descriptor area in a server row descriptor to be valid is if TYPE indicates an implementation-defined data type.

18) Insert this list element Subclause 21.5, "Implicit EXECUTE USING and OPEN USING clauses":

a) Let *NIDAL* be the number of item descriptor areas in *WPD* for which LEVEL is 0 (zero). If NIDAL is greater than *D*, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

b) There may be an implementation-defined conversion from type *SDT* to type *TDT*.

c) There may be an implementation-defined conversion from type *SDT* to type *UDT*.

19) Insert this list element Subclause 21.6, "Implicit CALL USING clause":

a) If the result is a zero-length character string, then it is implementation-defined whether or not an exception condition is raised: *data exception — zero-length character string*.

b) There may be an implementation-defined conversion from type *SDT* to type *TDT*.

20) Insert this list element Subclause 21.7, "Implicit FETCH USING clause":

a) If separate fetches for the same bound target are inconsistent in whether a locator is used, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

b) If the result is a zero-length character string, then it is implementation-defined whether or not an exception condition is raised: *data exception — zero-length character string*.

c) There may be an implementation-defined conversion from type *SDT* to type *TDT*.

21) Insert this list element Subclause 21.8, "Character string retrieval":

a) The null character that terminates C character strings is implementation-defined.

22) Insert this list element Subclause 22.1, "<foreign-data wrapper routine>":

a) It is implementation-defined which of the invocation of *WP* or *WF* is supported.

23) Insert this list element Subclause 22.2, "<foreign-data wrapper routine> invocation":

a) If the value of any input argument provided by *CP* falls outside the set of allowed values of the data type of the parameter, or if the value of any output argument resulting from the execution of the <foreign-data wrapper routine> falls outside the set of values supported by *CP* for that parameter, then the effect is implementation-defined.

b) If *RN* did not execute successfully, then one or more exception conditions may be raised as determined by implementation-defined rules.

24) Insert this list element Subclause 22.4, "AllocWrapperEnv":

a) If the resources to manage an foreign-data wrapper environment cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised.

25) Insert this list element Subclause 22.6, "ConnectForeignServer":

a) If the resources to manage an FS-connection cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised.

26) Insert this list element Subclause 22.16, "GetDescriptor":

a) If TYPE is 'HEADER', then header information from the descriptor area *D* is retrieved; if *FI* indicates an implementation-defined descriptor header field, then the value retrieved is the value of the implementation-defined descriptor header field identified by *FI*.

b) If TYPE is 'ITEM', then item information from the descriptor area *D* is retrieved; if *FI* indicates an implementation-defined descriptor item field, then the value retrieved is the value of the implementation-defined descriptor item field of *IDA* identified by *FI*.

27) Insert this list element Subclause 22.17, "GetDiagnostics":

a) If *TYPE* is 'HEADER' and *DI* indicates an implementation-defined diagnostics header field, then the value retrieved is the value of the implementation-defined diagnostics header field.

b) If *TYPE* is 'STATUS' and *DI* indicates an implementation-defined diagnostics header field, then the value retrieved is the value of the implementation-defined diagnostics header field.

c) If *TYPE* is 'STATUS' and *DI* indicates NATIVE_CODE, then the value retrieved is the implementation-defined native error code corresponding to the status condition.

d) If *TYPE* is 'STATUS' and *DI* indicates MESSAGE_TEXT, then the value retrieved is an implementation-defined character string.

e) If *TYPE* is 'STATUS' and *DI* indicates CLASS_ORIGIN, then the value retrieved shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined class value.

f) If *TYPE* is 'STATUS' and *DI* indicates SUBCLASS_ORIGIN, then the value retrieved shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined subclass value.

g) The null character that terminates C character strings is implementation-defined.

28) Insert this list element Subclause 22.33, "GetNumberOfServerOptions":

a) The maximum length of a variable-length character string is implementation-defined.

29) Insert this list element Subclause 22.42, "GetServerName":

a) The maximum length of a variable-length character string is implementation-defined.

30) Insert this list element Subclause 22.43, "GetServerOption":

a) The maximum length of a variable-length character string is implementation-defined.

31) Insert this list element Subclause 22.44, "GetServerOptionValueByName":

a) The maximum length of a variable-length character string is implementation-defined.

32) Insert this list element Subclause 22.45, "GetServerType":

a) The maximum length of a variable-length character string is implementation-defined.

33) Insert this list element Subclause 22.54, "GetTableReferenceTableName":

a) The maximum length of a variable-length character string is implementation-defined.

34) Insert this list element Subclause 22.57, "GetValueExpressionColumnName":

a) The maximum length of a variable-length character string is implementation-defined.

35) Insert this list element Subclause 22.62, "InitForeignRequest":

a) If the resources to manage an FDW-reply cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised.

b) If the resources to manage an FDW-execution cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised.

36) Insert this list element Subclause 22.63, "Iterate":

a) If the resources to manage an FDW-data cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised.

37) Insert this list element Subclause 22.67, "SetDescriptor":

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

b) If *FI* indicates TYPE and *V* indicates FLOAT, then the PRECISION field of *IDA* is set to the implementation-defined default value for the precision of the FLOAT data type.

c) The null character that terminates C character strings is implementation-defined.

d) If *FI* indicates TYPE and *V* indicates SMALLINT or INTEGER, then the SCALE field of *IDA* is set to 0 (zero) and the PRECISION field of *IDA* is set to the implementation-defined value for the precision of the SMALLINT or INTEGER data types, respectively.

e) If *FI* indicates TYPE and *V* indicates REAL or DOUBLE PRECISION, then the PRECISION field of *IDA* is set to the implementation-defined value for the precision of the REAL or DOUBLE PRECISION data types, respectively.

f) If *FI* indicates TYPE and *V* indicates an implementation-defined data type, then an implementation-defined set of fields of *IDA* are set to implementation-defined default values.

38) Insert this list element Table 5, "Fields in SQL/MED descriptor areas":

a) The maximum lengths of SQL/MED item descriptor fields whose data type is CHARACTER VARYING are implementation-defined.

b) SQL/MED supports implementation-defined header fields and implementation-defined item fields in row and parameter descriptor areas.

39) Insert this list element Table 36, "Ability to set SQL/MED descriptor fields":

a) 'ID' means that it is implementation-defined whether or not the descriptor field is settable.

b) SQL/MED supports implementation-defined descriptor header fields and implementation-defined descriptor item fields.

40) Insert this list element Table 35, "Ability to retrieve SQL/MED descriptor fields":

a) 'ID' means that it is implementation-defined whether or not the descriptor field is retrievable.

b) SQL/MED supports implementation-defined descriptor header fields and implementation-defined descriptor item fields.

41) Insert this list element Table 37, "SQL/MED descriptor field default values":

a) 'ID' means that the descriptor field's default value is implementation-defined.

b) SQL/MED supports implementation-defined descriptor header fields and implementation-defined descriptor item fields.

42) Insert this list element Table 33, "Codes used for SQL/MED descriptor fields":

a) SQL/MED supports implementation-defined descriptor header fields and implementation-defined descriptor item fields.

Annex C
(informative)

Implementation-dependent elements

This Annex references those places where this part of ISO/IEC 9075 states explicitly that the actions of a conforming implementation are implementation-dependent.

The term *implementation-dependent* is used to identify characteristics that may differ between implementations, but that are not necessarily specified for any particular implementation.

1) Insert this list element Subclause 4.14.3, "Return codes":

   a) After the execution of an SQL/MED routine, the values of all output arguments not explicitly defined by this part of ISO/IEC 9075 are implementation-dependent.

2) Insert this list element Subclause 4.14.4, "Diagnostics areas in SQL/MED":

   a) If multiple status records are generated, then the order in which status records are placed in a diagnostics area is implementation-dependent, with two exceptions.

3) Insert this list element Subclause 4.7, "Datalinks":

   a) The mechanism by which the SQL-server controls access and maintains integrity of external data sources is implementation-dependent. This mechanism is called the *datalinker*.

   b) The generation of the access token and the method of combining it with File Reference are implementation-dependent. Insert this list element Subclause 6.3, "<string value function>":

   a) The generation of the access token and the method of combining it with File Reference or the <hpath> or <fpath> of a File Reference are implementation-dependent.

4) Insert this list element Subclause 6.4, "<datalink value function>":

   a) The representation of the result of invoking a <datalink value constructor> is implementation-dependent.

5) Insert this list element Subclause 21.3, "Implicit DESCRIBE INPUT USING clause":

   a) If *D* is not zero, then those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values.

   b) If POPULATE WPD is true and *D* is not zero, then those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values and NAME is set to an implementation-dependent value.

6) Insert this list element Subclause 21.4, "Implicit DESCRIBE OUTPUT USING clause":

a) If *D* is not zero and the column name is implementation-dependent, then NAME is set to the implementation-dependent name of the column and UNNAMED is set to 1 (one).

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

7) Insert this list element Subclause 21.6, "Implicit CALL USING clause":

a) If *TDT* is a locator type and *SV* is not the null value, then a locator *L* that uniquely identifies *SV* is generated and the value *TV* of the *i*-th bound target is set to an implementation-dependent four-octet value that represents *L*.

b) If TYPE indicates ROW, *TV* is the null value, and *IP* is neither a null pointer for *IDA* nor for any of the subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose type indicates ARRAY or ARRAY_LOCATOR, then the value of the host variable addressed by *IP* for *IDA*, and that in all subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose TYPE indicates ARRAY or ARRAY_LOCATOR, is set to the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of variables addressed by *DP* and *LP* are implementation-dependent.

c) If TYPE does not indicate ROW, *TV* is the null value, and *IP* is not a null pointer, then the value of the host variable addressed by *IP* is set to the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of the host variables addressed by *DP* and *LP* are implementation-dependent.

8) Insert this list element Subclause 21.7, "Implicit FETCH USING clause":

a) If TDT is a locator type and *SV* is not the null value, then a locator *L* that uniquely identifies *SV* is generated and the value *TV* of the *i*-th bound target is set to an implementation-dependent four-octet value that represents *L*.

b) If TYPE indicates ROW, *TV* is the null value, and *IPE* is not a null pointer for *IDA* nor for any of the subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose type indicates ARRAY or ARRAY_LOCATOR, then the value of the host variable addressed by *IPE* for *IDA*, and that in all subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose TYPE indicates ARRAY or ARRAY_LOCATOR, is set to the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of variables addressed by *DPE* and *LPE* are implementation-dependent.

c) If TYPE does not indicate ROW, *TV* is the null value, and *IPE* is not a null pointer, then the value of the host variable addressed by *IPE* is set to the appropriate 'Code' for SQL NULL DATA in Table 26, "Miscellaneous codes used in SQL/CLI", in ISO/IEC 9075-3, and the values of the host variables addressed by *DPE* and *LPE* are implementation-dependent.

9) Insert this list element Subclause 21.9, "Binary large object string retrieval":

a) If *TT* indicates BINARY LARGE OBJECT and *L* is not greater than *TL*, then the first *L* octets of *T* are set to *V* and the values of the remaining octets of *T* are implementation-dependent. Insert this list element Subclause 21.8, "Character string retrieval":

a) If null termination is **false** for the SQL-server and *L* is not greater than *TL*, then the first *L* octets of *T* are set to *V* and the values of the remaining octets of *T* are implementation-dependent.

b) If null termination is true for the SQL-server and *L* is not greater than *TL*, then the first (*L*+*NB*) octets of *T* are set to *V* concatenated with a single implementation-defined null character that terminates a C character string and the values of the remaining characters of *T* are implementation-dependent.

Insert this list element Subclause 22.17, "GetDiagnostics":

a) If null termination is **false** for the SQL-server and *L* is not greater than *BL*, then the first *L* octets of DiagInfo are set to *V* and the values of the remaining octets of DiagInfo are implementation-dependent.

b) If null termination is **true** for the SQL-server and *L* is not greater than *BL*, then the first (*L*+*k*) octets of DiagInfo are set to *V* concatenated with a single implementation-defined null character that terminates a C character string and the values of the remaining characters of DiagInfo are implementation-dependent.

10) Insert this list element Subclause 22.62, "InitForeignRequest":

a) If *P* is a <dynamic select statement> or a <dynamic single row select statement>, then a unique implementation-dependent name becomes the cursor name associated with FDW-execution.

11) Insert this list element Subclause 22.63, "Iterate":

a) If an exception condition is raised during the derivation of any target value, then the values of all the bound targets are implementation-dependent and *CR* remains positioned on the current row.

12) Insert this list element Subclause 22.67, "SetDescriptor":

a) If *FI* indicates TYPE, then all fields of *IDA* other than those prescribed are set to implementation-dependent values.

b) If *FI* indicates DATETIME_INTERVAL_CODE and the TYPE field of *IDA* indicates a <datetime type>, then all the fields of *IDA* other than DATETIME_INTERVAL_CODE and TYPE are set to implementation-dependent values.

c) If an exception condition is raised, then the field of *IDA* indicated by *FI* is set to an implementation-dependent value.

Annex D
(informative)

Deprecated features

It is intended that the following features will be removed at a later date from a revised version of ISO/IEC 9075:

No additional deprecated items.

Annex E
(informative)

Incompatibilities with ISO/IEC 9075:1999

This part of ISO/IEC 9075 introduces some incompatibilities with the earlier version of Database Language SQL as specified in ISO/IEC 9075:1999. Unless specified in this Annex, features and capabilities of Database Language SQL are compatible with the earlier version of ISO/IEC 9075:1999.

1) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:

   — DATALINK

Annex F
(informative)

Typical header files

F.1 C Header File SQLCLI.H

```
/* API declaration data types                                    */
typedef unsigned char   SQLDATALINK;

/* datalink attributes                                           */
#define SQL_ATTR_DL_COMMENT                      1
#define SQL_ATTR_DL_TYPE                         2
#define SQL_ATTR_DL_URL_COMPLETE                 3
#define SQL_ATTR_DL_URL_PATH                     4
#define SQL_ATTR_DL_URL_PATH_ONLY                5
#define SQL_ATTR_DL_URL_SCHEME                   6
#define SQL_ATTR_DL_URL_SERVER                   7

/* SQL data type codes                                           */
#define SQL_DATALINK                            50

/*      GetFunctions values to identify CLI routines             */
#define SQL_API_SQLBUILDDATALINK                1029
#define SQL_API_SQLGETDATALINKATTR              1034

/*      Information requested by GetInfo( )                      */
#define SQL_MAXIMUM_DATALINK_LENGTH            20004



/* Function prototypes                                           */

SQLRETURN  SQLBuildDataLink(SQLHSTMT *StatementHandle,
           SQLCHAR *LinkType, SQLINTEGER LinkTypeLength,
           SQLCHAR *DataLocation, SQLINTEGER DataLocationLength,
           SQLCHAR *Comment, SQLINTEGER CommentLength,
           SQLCHAR *DataLink, SQLINTEGER DataLinkLength,
           SQLINTEGER *StringLength);

SQLRETURN  SQLGetDataLinkAttr(SQLHSTMT *StatementHandle,
           SQLSMALLINT Attribute,
           SQLCHAR *DataLink, SQLINTEGER DataLinkLength,
           SQLPOINTER Value, SQLINTEGER BufferLength,
           SQLINTEGER *StringLength);
```

F.2 COBOL Library Item SQLCLI

```
* DATALINK ATTRIBUTE
01 SQL-ATTR-DL-COMMENT                 PIC S9(4) BINARY VALUE IS     1.
01 SQL-ATTR-DL-PATH                    PIC S9(4) BINARY VALUE IS     2.
01 SQL-ATTR-DL-URL-COMPLETE            PIC S9(4) BINARY VALUE IS     3.
01 SQL-ATTR-DL-URL-PATH                PIC S9(4) BINARY VALUE IS     4.
01 SQL-ATTR-DL-URL-PATH-ONLY           PIC S9(4) BINARY VALUE IS     5.
01 SQL-ATTR-DL-URL-SCHEME              PIC S9(4) BINARY VALUE IS     6.
01 SQL-ATTR-DL-URL-SERVER              PIC S9(4) BINARY VALUE IS     7.

* SQL DATA TYPE CODES
01 SQL-DATALINK                        PIC S9(4) BINARY VALUE IS    50.

* SQLRGETFUNCTIONS VALUES TO IDENTIFY CLI ROUTINES
01 SQL-API-SQLBUILDDATALINK            PIC S9(4) BINARY VALUE IS  1029.
01 SQL-API-SQLGETDATALINKATTR          PIC S9(4) BINARY VALUE IS  1034.

* INFORMATION REQUESTED BY SQLRGETINFO
01 SQL-MAXIMUM-DATALINK-LENGTH         PIC S9(4) BINARY VALUE IS 20004.
```

Annex G
(informative)

SQL feature and package taxonomy

This Annex describes a taxonomy of features and packages defined in this part of ISO/IEC 9075.

Table 40, "SQL/MED feature taxonomy for features outside Core SQL", contains a taxonomy of the features of the SQL language not in Core SQL that are specified in this part of ISO/IEC 9075.

In this table, the first column contains a counter that may be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to ISO/IEC 9075 without notice.

The second column, "Feature ID", specifies the formal identification of each feature and each subfeature contained in the table. The Feature ID is stable and can be depended on to remain constant. A Feature ID value comprises either a letter and three digits or a letter, three digits, a hyphen, and one or two additional digits. Feature ID values containing a hyphen and additional digits indicate "subfeatures" that help to define complete features, which are in turn indicated by Feature ID values without a hyphen. Only entire features are used to specify the contents of Core SQL and various packages.

The "Feature Description" column contains a brief description of the feature or subfeature associated with the Feature ID value.

Table 40—SQL/MED feature taxonomy for features outside Core SQL

	Feature ID	Feature Name
1	M001	Datalinks
2	M002	Foreign-data wrappers
3	M003	Datalinks via SQL/CLI
4	M004	Datalinks via Embedded SQL

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined; index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

— D —

— G —

— H —

— I —

— K —