INTERNATIONAL STANDARD

ISO/IEC 9075-9

Fourth edition
2016-12-15

Information technology — Database languages — SQL —

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

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

Partie 9: Gestion des données externes (SQL/MED)

Reference number
ISO/IEC 9075-9:2016(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Tables

Table

Page

Figures

Figure

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, SC 32, *Data management and interchange*.

This fourth edition of ISO/IEC 9075-9 cancels and replaces the third edition (ISO/IEC 9075-9:2008), which has been technically revised. It also incorporates Technical Corrigendum ISO/IEC 9075-9:2008/Cor.1:2010.

A list of all parts in the ISO/IEC 9075 series, published under the general title *Information technology — Database languages — SQL*, can be found on the ISO website.

NOTE The individual parts of multi-part standards are not necessarily published together. New editions of one or more parts can be published without publication of new editions of other parts.

Introduction

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

1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 9075.

2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of ISO/IEC 9075, constitute provisions of this part of ISO/IEC 9075.

3) Clause 3, “Definitions, notations, and conventions”, defines the notations and conventions used in this part of ISO/IEC 9075.

4) Clause 4, “Concepts”, presents concepts 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, “URLs”, specifies the format of URLs used in this part of ISO/IEC 9075.

9) Clause 9, “Additional common rules”, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.

10) Clause 10, “Additional common elements”, defines additional common elements used in the definition of foreign tables, foreign servers, and foreign-data wrappers.

11) Clause 11, “Schema definition and manipulation”, defines facilities related to foreign tables and datalink type support for creating and managing a schema.

12) Clause 12, “Catalog manipulation”, defines facilities for creating, altering, and dropping foreign servers and foreign-data wrappers.

13) Clause 13, “Access control”, defines facilities for controlling access to SQL-data.

14) Clause 14, “SQL-client modules”, defines SQL-client modules and externally-invoked procedures.

15) Clause 15, “Additional data manipulation rules”, defines additional rules for data manipulation.

16) Clause 16, “Session management”, defines the SQL-session management 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 21, “SQL/MED common specifications”, specifies common facilities used by SQL/MED.

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

23) Clause 23, “Diagnostics management”, defines the diagnostics management facilities.

24) Clause 24, “Information Schema”, defines viewed tables that contain schema information.

25) Clause 25, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.

26) Clause 26, “Status codes”, defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

27) Clause 27, “Conformance”, specifies the way in which conformance to this part of ISO/IEC 9075 may be claimed.

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

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

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

31) 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.

32) Annex E, “Incompatibilities with ISO/IEC 9075:2008”, is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 9075.

33) Annex F, “SQL feature taxonomy”, is an informative Annex. It identifies features 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.

34) Annex G, “Defect reports not addressed in this edition of this part of ISO/IEC 9075”, is an informative Annex. It describes the Defect Reports that were known at the time of publication of this part of this International Standard. Each of these problems is a problem carried forward from the previous edition of ISO/IEC 9075. No new problems have been created in the drafting of this edition of this International Standard.

35) Annex H, “Typical header files”, is an informative Annex. It provides examples of typical definition files for application programs using the SQL Call-Level Interface.

36) Annex I, “SQL/MED model”, is an informative Annex. It uses annotated diagrams to illustrate the more important concepts of the model of SQL/MED, including the relationships between the SQL-server, foreign-data wrappers, and foreign servers.

In the text of this part of ISO/IEC 9075, Clauses and Annexes begin new odd-numbered pages, and in Clause 5, “Lexical elements”, through Clause 27, “Conformance”, Subclauses begin new pages. Any resulting blank space is not significant.

(Blank page)

Information technology — Database languages — SQL —

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

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-data wrappers and datalink types.

(Blank page)

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 ISO and IEC standards

[ISO9075-1] ISO/IEC 9075-1:2016, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

[ISO9075-2] ISO/IEC 9075-2:2016, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

[ISO9075-3] ISO/IEC 9075-3:2016, *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*.

[ISO9075-11] ISO/IEC 9075-11:2016, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*.

2.2 Other international standards

[RFC2368] RFC 2368, *The mailto URL scheme*, R. Hoffman, L. Masinter, J. Zawinski.
http://www.ietf.org/rfc/rfc2368.txt

[RFC3986] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter.
http://www.ietf.org/rfc/rfc3986.txt

[XML] is used to reference either [XML 1.0] or [XML 1.1] when there is no significant difference between the two for the purposes of a given citation.

[XML 1.0] *(Recommendation) Extensible Markup Language (XML) Version 1.0*.
http://www.w3.org/TR/xml

[XML 1.1] *(Recommendation) Extensible Markup Language (XML) Version 1.1*.
http://www.w3.org/TR/xml11

(Blank page)

3 Definitions, notations, and conventions

3.1 Definitions

This Subclause modifies Subclause 3.1, "Definitions", in ISO/IEC 9075-2.

3.1.1 Definitions taken from XML

For the purposes of this document, the definitions of the following terms given in [XML] apply:

3.1.1.1 Valid XML document

3.1.1.2 XML document

3.1.1.3 XML document type declaration ="(also known as a DTD)

3.1.2 Definitions provided in Part 9

For the purposes of this document, in addition to those definitions taken from other sources, the following definitions apply:

3.1.2.1 access token
encrypted value returned under certain conditions by an SQL-server in combination with the File Reference of a datalink value

NOTE 2 — An access token is either a *read token* or a *write token*.

3.1.2.2 datalink
value, of datatype DATALINK, referencing some file that is not part of the SQL-environment

NOTE 3 — The file is assumed to be managed by some external file manager.

3.1.2.3 datalinker
implementation-dependent component for enabling integrity control, recovery, and access control for external files

3.1.2.4 external data
data that is not managed by an SQL-server involved in an SQL-session, but that is nevertheless accessible to that SQL-session

3.1.2.5 foreign-data wrapper
named collection of routines, invocable by the SQL-server, supporting the programming interface specified for such routines in this part of ISO/IEC 9075

3.1.2.6 foreign server

named server, external to the SQL-environment, but known to the SQL-server, that manages external data

3.1.2.7 foreign server request
statement that an SQL-server submits to a foreign-data wrapper

3.1.2.8 foreign table
named table whose rows are supplied when needed by some foreign server

NOTE 4 — The mechanism by which these rows are supplied is provided by a foreign-data wrapper. The data constituting a foreign table is not part of the SQL-environment.

3.1.2.9 integrity control option
link control option specifying the level of integrity of the link between a datalink and the file that it references

3.1.2.10 link control
property of a column of data type DATALINK, specifying the extent to which the links between datalinks in that column and the files they reference are to be monitored (in various specific manners)

3.1.2.11 read permission option
link control option specifying how permission to read external files referenced by certain datalinks is determined

3.1.2.12 recovery option
link control option specifying whether or not point in time recovery is required for the files referenced by certain datalinks

3.1.2.13 routine mapping
implementation-defined mapping of an SQL-invoked routine to an equivalent concept maintained by a foreign server

3.1.2.14 SQL/MED-implementation
SQL-implementation that processes SQL-statements that are possibly extended by the language defined in this part of ISO/IEC 9075

NOTE 5 — A *conforming SQL/MED-implementation* is an SQL/MED-implementation that satisfied the requirements for SQL/MED-implementations as defined in Clause 27, "Conformance".

3.1.2.15 unlink option
link control option specifying the action to be taken when certain sites occupied by datalinks are updated or deleted

3.1.2.16 user mapping
implementation-defined mapping of an authorization identifier to an equivalent concept maintained by a foreign server

3.1.2.17 write permission option
link control option specifying how permission to write files referenced by certain datalinks is determined

4 Concepts

This Clause modifies Clause 4, "Concepts", in ISO/IEC 9075-2.

4.1 Data types

This Subclause modifies Subclause 4.1, "Data types", in ISO/IEC 9075-2.

4.1.1 Naming of predefined types

This Subclause modifies Subclause 4.1.2, "Naming of predefined types", in ISO/IEC 9075-2.

Insert after 1st paragraph SQL defines a predefined data type named by the following <key word>: DATALINK.

Insert after 3rd paragraph For reference purposes, the data type DATALINK is referred to as a (or the) *datalink type*.

4.1.2 Data type terminology

This Subclause modifies Subclause 4.1.4, "Data type terminology", in ISO/IEC 9075-2.

Augment the list in the 11th paragraph

— A type *T* is *DATALINK-ordered* if *T* is *S*-ordered, where *S* is the set of datalink types.

4.2 Foreign servers

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

A *foreign server descriptor* is a catalog element, identified by a *foreign server name* and created by invoking a <foreign server definition>. A foreign server descriptor consists of:

— A foreign server name, identifying the foreign server locally to the SQL-server.

— The authorization identifier of the owner of the foreign server descriptor.

— The name of the foreign-data wrapper.

— A generic options descriptor.

— Optionally, the foreign server type.

— Optionally, the foreign server version.

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

A foreign server descriptor is said to be *owned by* or to have been *created by* the current authorization identifier for the SQL-session when the <foreign server definition> was invoked.

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 the ability to process a subset of statements conforming to ISO/IEC 9075, particularly the statements comprising Feature E051, "Basic query specification", in a standard-conforming manner. 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 provides 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 6 — 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.

Some foreign servers, especially SQL-aware foreign servers, admit the concept of a schema and the concept of a table that are similar to SQL-schemas and to base tables, respectively. Such servers may (and SQL-aware foreign servers do) maintain schema information about those entities, such as the Information Schema and Definition Schema specified in [ISO9075-11].

If a foreign server maintains schema information about entities analogous to SQL-schemas and base tables, then execution of an <import foreign schema statement> retrieves information about the tables (either all or only some, as specified in the <import foreign schema statement>) associated with the named SQL-schema analog and effectively performs one or more <foreign table definition> statement executions.

If a foreign server does not maintain such information or does not admit the concept of a schema, then foreign tables managed by that server shall be specified by means of explicit <foreign table definition>s.

This International Standard does not specify the manner in which the SQL-server and the foreign-data wrapper interact to cause information about foreign tables to be retrieved by execution of an <import foreign schema statement>. In particular, no foreign-data wrapper interface routines are specified to support such interaction. Such interaction is implementation-dependent.

4.3 Foreign-data wrappers

A foreign-data wrapper is the mechanism by which the SQL-server accesses external data managed by foreign servers. Every foreign server is accessed through exactly one foreign-data wrapper, but one foreign-data wrapper can be used to access several different foreign servers. A foreign-data wrapper is made up of foreign-data wrapper interface routines and a set of routines written in a programming language. Foreign-data wrapper interface routines are used to access every foreign server whose descriptor includes the name of that foreign-data wrapper. It is possible for a foreign-data wrapper to exist that is not used to access any foreign server.

A *foreign-data wrapper descriptor* is a catalog element, identified by a *foreign-data wrapper name* and created by invoking a <foreign-data wrapper definition>. A <foreign-data wrapper definition> specifies the foreign-data wrapper name, a library name that identifies a library containing the foreign-data wrapper interface routines, and the name of the language in which the foreign-data wrapper interface routines are written.

A foreign-data wrapper descriptor consists of:

— A foreign-data wrapper name.

— The authorization identifier of the owner of the foreign-data wrapper descriptor.

— The name of the language in which the foreign-data wrapper interface routines are written.

— A generic options descriptor.

— A library name.

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

4.4 User mappings

A user mapping is an SQL-environment element, pairing an authorization identifier *U* or the special identifier PUBLIC, denoting all <authorization identifier>s in the SQL-environment, 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 when the current authorization identifier is *U*. The mapping is specified by generic options defined by the foreign-data wrapper.

A user mapping is defined by invoking a <user mapping definition>. Invocation of a <user mapping definition> results in the creation of a user mapping descriptor in the SQL-environment. 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>.

4.5 Routine mappings

A routine mapping is an SQL-environment element, pairing an SQL-invoked routine *SIR* with a foreign server *FS*. It defines how to map *SIR* to an equivalent concept known to *FS* when a foreign table *FT* whose source is *FS* is to be accessed and the foreign server request that includes *FT* also includes a reference to *SIR*. The mapping is specified by generic options defined by the foreign-data wrapper.

A routine mapping is defined by invoking a <routine mapping definition>. Invocation of a <routine mapping definition> results in the creation of a routine mapping descriptor in the SQL-environment. A routine mapping descriptor consists of:

— The name of the routine mapping.

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

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

— A generic options descriptor.

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

4.6 Generic options

Several of the objects used in connection with external data support the specification of *generic options*. These objects are foreign-data wrappers, foreign servers, foreign tables, columns of foreign tables, user mappings, and routine mappings. A generic option is an option name paired with an optional option value. Both the option name and the permissible ranges of option values of a generic option are defined by the foreign-data wrappers. 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. The generic options are stored in the SQL-server for the foreign-data wrapper to retrieve when the foreign-data wrapper needs this information.

Generic options may be specified in either <foreign-data wrapper definition>, <foreign server definition>, <foreign table definition>, <user mapping definition>, <routine mapping definition>, <alter foreign-data wrapper statement>, <alter foreign server statement>, <alter foreign table statement>, <alter user mapping statement>, or <alter routine mapping statement>.

Generic options are specific to the object for which they are defined. For example, the generic options for a foreign table are most likely different from the generic options for a foreign server, in both option names and option values. Furthermore, generic options are highly dependent on the foreign-data wrapper that is used to access the external data. For example, the generic options for a foreign server that uses a foreign-data wrapper *A* might be totally different from the generic options specified for another foreign server that uses a foreign-data wrapper *B*. Even the fact that the option names of two generic options for two different foreign-data wrappers might be the same does not necessarily mean that the semantics and therefore the permissible ranges of option values are the same.

Since an SQL-server cannot anticipate the different kinds of foreign-data wrappers with which it is likely to deal, no generic option can ever be determined by the SQL-server or by this part of ISO/IEC 9075. Only a foreign-data wrapper can specify generic options for that foreign-data wrapper, or for a foreign server, a foreign table, a column of a foreign table, a user mapping, or a routine mapping for which it is used.

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.

4.7 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 `GetOpts()` 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 `GetOpts()` 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 may comprise an XML document or that may be returned in a format defined by the foreign-data wrapper. If the contents comprise an XML document, then it shall be a valid XML document, the format of which is specified by a Document Type Declaration (DTD) that is either internal to the XML document or external (requiring that it be available to the SQL-server in an implementation-defined manner).

NOTE 7 — This edition of this part of ISO/IEC 9075 specifies the use of a DTD. Future editions may specify the use of an XML Schema, either as an alternative to a DTD or instead of a DTD.

4.8 Datalinks

A datalink is a value of the DATALINK data type. A datalink references some file that is not part of the SQL-environment. The file is assumed to be managed by some external file manager. A datalink is conceptually represented by:

— File Reference: A character string forming a reference to an external file.

— SQL-Mediated Read Access Indication: A boolean value, where *True*, in datalink *DL* indicates that the referenced file, being linked to the SQL-environment, is accessible to be read only by use of the specially provided operations (see below) on *DL*.

— SQL-Mediated Write Access Indication: A boolean value, where *True*, in datalink *DL* indicates that the referenced file, being linked to the SQL-environment, is accessible to be modified only by use of the specially provided operations (see below) on *DL*.

— Write Token: An implementation-dependent value that represents an access token that is used to read or modify the File Reference. This value can be the null value.

— Construction Indication: A character string indicating how the datalink was constructed. Possible values are: NEWCOPY, PREVIOUSCOPY, and the null value.

The File Reference of a datalink is accessible by invoking operators defined in this part of ISO/IEC 9075. The character set of the File Reference, referred to as the *datalink character set* is implementation-defined.

The purpose of datalinks is to provide a mechanism to synchronize the integrity control, recovery, and access control of the files 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 files identified by datalink values. The mechanisms that enable integrity control, recovery, and access control for the files represented by the datalink values are implementation-dependent. These mechanisms are collectively called the *datalinker*.

A file is *linked* to the SQL-environment whenever execution of an SQL-data change statement causes a value *DL1* that references that file to appear in some datalink column whose descriptor includes the link control FILE LINK CONTROL. If the read permission option included in the column descriptor is DB, then access to the

referenced file is said to be *SQL-mediated*. This is indicated by setting the SQL-Mediated Read Access Indication of *DL1* to *True*, and *DL1* is said to be an *SQL-mediated datalink*. If the read permission option included in the column descriptor is not DB, then the SQL-Mediated Read Access Indication of *DL1* is set to *False*. If the write permission option included in the column descriptor is ADMIN, then this is indicated by setting the SQL-Mediated Write Access Indication of *DL1* to *True*. If the write permission option included in the column descriptor is not ADMIN, then the SQL-Mediated Write Access Indication of *DL1* is set to *False*.

Execution of an SQL-data change statement that causes a value *DL2* to appear in a datalink column defined with the link control NO LINK CONTROL does not cause any file to be linked to the SQL-environment.

A linked file cannot be renamed or deleted by any agency outside of the SQL-environment. A datalink value always references just one file. A file is *unlinked* from the SQL-environment whenever execution of an SQL-data change statement causes a datalink that references that file 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 file might be deleted, or the datalinker might return control of the file to the external data manager.

With the function provided by datalinks and the datalinker, it is possible to specify that access to the files should be mediated by the SQL-server rather than by the external data manager. When access to the files is mediated by an SQL-server, any request to access a file shall operate on an SQL-mediated datalink to obtain a character string with which to reference the file, using one of the operators 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*. An access token is either a *read token* or a *write token*, depending on the function that is used to construct the character string. 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 a file, the datalinker checks to see if the access token is *valid*. If it is valid, then the application is allowed to access the file pointed to by the File Reference. Every attempt by an application to access, without a valid access token, a file referenced by an SQL-mediated datalink is unsuccessful. The time at which a valid access token ceases to be valid is implementation-defined.

The content of an SQL-mediated file cannot be modified, unless the SQL-Mediated Write Access Indication of the datalink value *DL* referencing this file is *True*. After an application has modified the file, it uses a <datalink value constructor> that specifies either DLNEWCOPY or DLPREVIOUSCOPY to construct a new datalink value. This new datalink value is then used to update the site that contains *DL*.

NOTE 8 — Updating the site that contains a datalink in the manner described here is called “update-in-place”.

Datalinks are not comparable. A datalink is assignable only to sites of type DATALINK.

A datalink data type is described by a *datalink data type descriptor*. A datalink data type descriptor consists of the name DATALINK and the set of *link control options*:

— 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, ADMIN, or BLOCKED). If the write permission option is ADMIN, then additionally the access token indication (either NOT REQUIRING TOKEN FOR UPDATE or REQUIRING TOKEN FOR UPDATE).

— The recovery option (NO or YES).

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

The meanings of the various link control options are:

— NO LINK CONTROL: Although every File Reference shall conform to the Format and Syntax Rules of Subclause 8.1, "URL format", it is permitted for there to be no file referenced by that File Reference. 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 Reference shall reference an existing file. Further file control depends on the link control options.

— INTEGRITY ALL: Files referenced by File References 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: Files referenced by File References can be deleted or renamed using operators provided by the file manager, unless a datalinker is installed in connection with the file manager.

— INTEGRITY NONE: Files referenced by File References can only be deleted or renamed using operators provided by the file manager. This option is not available if FILE LINK CONTROL is specified.

— READ PERMISSION FS: Permission to read files referenced by datalinks is determined by the file manager.

— READ PERMISSION DB: Datalinks of this type are SQL-Mediated. That is to say, permission to read files referenced by such datalinks is determined by the SQL-implementation.

— WRITE PERMISSION FS: Permission to write files referenced by datalinks is determined by the file manager.

— WRITE PERMISSION ADMIN REQUIRING TOKEN FOR UPDATE: Permission to write files referenced by datalinks is determined by the SQL-implementation and the datalinker. This option is only available if READ PERMISSION DB is also specified. If a site that was declared with this write permission is updated, then the access token used to open and modify the file is required to be contained in the file reference specified in the invocation of the functions DLNEWCOPY or DLPREVIOUSCOPY that yield the value with which the site is updated.

— WRITE PERMISSION ADMIN NOT REQUIRING TOKEN FOR UPDATE: Permission to write files referenced by datalinks is determined by the SQL-implementation and the datalinker. This option is only available if READ PERMISSION DB is also specified. If a site that was declared with this write permission is updated, then an access token is not required to be contained in the file reference specified in the invocation of the functions DLNEWCOPY or DLPREVIOUSCOPY that yield the value with which the site is updated.

— WRITE PERMISSION BLOCKED: Write access to files 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 files referenced by datalinks.

NOTE 9 — "point in time recovery" is an implementation-defined mechanism that provides for recovery that is coordinated between the SQL-server and the files of external file manager referenced by datalinks.

— RECOVERY NO: Point in time recovery of files referenced by datalinks is disabled.

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

— ON UNLINK DELETE: A file referenced by a datalink is deleted when it is unlinked.

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

Table 1, "Valid datalink file control options", specifies what combinations of datalink file control options are allowed.

Table 1 — 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
ALL	DB	BLOCKED	YES	RESTORE
ALL	DB	BLOCKED	YES	DELETE
ALL	DB	ADMIN	NO	RESTORE
ALL	DB	ADMIN	NO	DELETE
ALL	DB	ADMIN	YES	RESTORE
ALL	DB	ADMIN	YES	DELETE
SELECTIVE	FS	FS	NO	NONE

NOTE 10 — In Table 1, "Valid datalink file control options", the write permission option ADMIN is an abbreviation for both ADMIN REQUIRING TOKEN FOR UPDATE and ADMIN NOT REQUIRING TOKEN FOR UPDATE.

The default value of a site whose declared type is DATALINK is the null value. Datalinks are subject to certain restrictions. As a consequence of these restrictions, neither datalinks nor expressions whose declared type is DATALINK-ordered can appear in (among other places):

— <comparison predicate>.

— <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.

— <select list> of an operand of UNION, INTERSECT, and EXCEPT.

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

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.

4.8.1 Operations involving datalinks

4.8.1.1 Operators that operate on datalinks

<url complete expression> returns the File Reference of a given datalink, possibly combined with a read token.

<url complete for write expression> returns the File Reference of a given datalink, possibly combined with a write token.

<url complete only expression> returns the File Reference, excluding any access token, of a given datalink.

<url path expression> returns the path, including any read token, of the File Reference of a given datalink.

<url path for write expression> returns the path, including any write token, of the File Reference of a given datalink.

<url path only expression> returns the path, excluding any access token, of the File Reference of a given datalink.

<url scheme expression> returns the scheme of the File Reference of a given datalink.

<url server expression> returns the host of the File Reference of a given datalink.

NOTE 11 — “host”, “scheme”, and “path” are defined in Subclause 6.6, “<datalink value function>”.

4.8.1.2 Other operators involving datalinks

A <datalink value constructor> specifies either DLVALUE, DLNEWCOPY, or DLPREVIOUSCOPY. DLVALUE returns a datalink value, given only a File Reference, returns the corresponding datalink. DLNEWCOPY and DLPREVIOUSCOPY return a datalink value, given a File Reference and an indication of whether the File Reference includes a write token.

The datalink value returned by DLNEWCOPY indicates to the SQL-server that the content of the file, referenced by that datalink, is different (*i.e.*, the content has changed, but not the URL) from what was previously referenced by the datalink.

The datalink value returned by DLPREVIOUSCOPY indicates to the SQL-server that the content of the file might have changed, but the application is not interested in maintaining the changed file. The original file is restored in an implementation-dependent fashion.

4.9 Columns, fields, and attributes

This Subclause modifies Subclause 4.13, "Columns, fields, and attributes", in ISO/IEC 9075-2.

Append this paragraph The term *constituent* is defined for values such that a value *V2* either is or is not a constituent of a value *V1*.

NOTE 12 — For example, the integer 2 and the character string 'one' are both constituents of the row value denoted by `ROW (2, 'one')`. By contrast, the integer 3 is not a constituent of that row value.

V2 is an *immediate constituent* of *V1* if any of the following are true:

— *V1* is a value of some predefined data type or of some distinct type whose source type is some predefined data type and *V2* is identical to *V1*.

— *V1* is a value of some structured type *ST* and, for some attribute *A* of *ST*, *V2* is identical to *V1.A*().

— *V1* is a value of some row type *RT* and, for some field *F* of *RT*, *V2* is identical to *V1.F*.

— *V1* is a value of some collection type *CT* and *V2* is an element of *V1*.

V2 is a *constituent* of *V1* if *V2* is an immediate constituent of *V1* or there is some value *V3* such that *V3* is an immediate constituent of *V1* and *V2* is a constituent of *V3*.

4.10 Tables

This Subclause modifies Subclause 4.15, "Tables", in ISO/IEC 9075-2.

4.10.1 Introduction to tables

This Subclause modifies Subclause 4.15.1, "Introduction to tables", in ISO/IEC 9075-2.

Add the following table type to the list of table types 3rd paragraph

— foreign table,

4.10.2 Base tables

This Subclause modifies Subclause 4.15.2, "Base tables", in ISO/IEC 9075-2.

4.10.2.1 Foreign tables

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").

4.10.3 Unique identification of tables

This Subclause modifies Subclause 4.15.5, "Unique identification of tables", in ISO/IEC 9075-2.

— Append after 4th list item The <table name> of an external table uniquely identifies a multiset of rows.

4.10.4 Table descriptors

This Subclause modifies Subclause 4.15.7, "Table descriptors", in ISO/IEC 9075-2.

Replace 1st 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 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.

— An indication of whether the foreign table is updatable or not.

NOTE 13 — This part of ISO/IEC 9075 currently restricts foreign tables such that they are neither insertable-into nor updatable. Future versions of this part of ISO/IEC 9075 may relax these restrictions.

4.10.5 Syntactic analysis of derived tables and cursors

This Subclause modifies Subclause 4.15.8, "Syntactic analysis of derived tables and cursors", in ISO/IEC 9075-2.

— Replace 1st list item of the 7th paragraph If *TORQN* identifies a base table or a foreign table, or if *TORQN* is a <transition table name>, then *TORQN* has no generally underlying table specifications.

4.11 Functional dependencies

This Subclause modifies Subclause 4.24, "Functional dependencies", in ISO/IEC 9075-2.

4.11.1 Overview of functional dependency rules and notations

This Subclause modifies Subclause 4.24.1, "Overview of functional dependency rules and notations", in ISO/IEC 9075-2.

Replace 1st paragraph This Subclause defines *functional dependency* and specifies a minimal set of rules that a conforming implementation shall follow to determine functional dependencies and candidate keys in base tables, foreign tables, and <query expression>s.

4.11.2 Known functional dependencies in a foreign table

There are no rules in this part of ISO/IEC 9075 to determine known functional dependencies in a foreign table. However, implementation-defined rules may determine known functional dependencies, if any, in a foreign table.

4.12 SQL-schemas

This Subclause modifies Subclause 4.26, "SQL-schemas", in ISO/IEC 9075-2.

Replace 5th paragraph Base tables, foreign tables, and views are identified by <table name>s. A <table name> consists of a <schema name> and an <identifier>. The <schema name> identifies the schema in which a persistent base table, foreign table, or view identified by the <table name> is defined. Base tables, foreign tables, and views defined in different schemas can have <identifier>s that are equal according to the General Rules of Subclause 8.2, "<comparison predicate>", in [ISO9075-2].

4.13 SQL-statements

This Subclause modifies Subclause 4.39, "SQL-statements", in ISO/IEC 9075-2.

4.13.1 SQL-statements classified by function

This Subclause modifies Subclause 4.39.2, "SQL-statements classified by function", in ISO/IEC 9075-2.

4.13.1.1 SQL-schema statements

This Subclause modifies Subclause 4.39.2.1, "SQL-schema statements", in ISO/IEC 9075-2.

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

— <import foreign schema statement>

— <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>
— <routine mapping definition>
— <alter routine mapping statement>
— <drop routine mapping statement>

4.13.1.2 SQL-session statements

This Subclause modifies Subclause 4.39.2.7, "SQL-session statements", in ISO/IEC 9075-2.

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

— <set passthrough statement>

4.14 Basic security model

This Subclause modifies Subclause 4.40, "Basic security model", in ISO/IEC 9075-2.

4.14.1 Privileges

This Subclause modifies Subclause 4.40.2, "Privileges", in ISO/IEC 9075-2.

Augment the list in the 1st paragraph

— foreign table
— foreign-data wrapper
— foreign server

NOTE 14 — Privileges granted on foreign tables are not privileges to use the data constituting foreign tables, but privileges to use the definitions of the foreign tables. The privileges to access the data constituting the foreign tables are enforced by the foreign server, based on the user mapping. Consequently, a request by an SQL-client to access external data may raise exceptions.

Augment the list in the 11th paragraph

— foreign-data wrapper

— foreign server

4.15 SQL-transactions

This Subclause modifies Subclause 4.41, "SQL-transactions", in ISO/IEC 9075-2.

4.15.1 Properties of SQL-transactions

This Subclause modifies Subclause 4.41.3, "Properties of SQL-transactions", in ISO/IEC 9075-2.

Augment the 2nd paragraph Add foreign tables to the list of objects for which the term *read-only* applies.

4.16 SQL-sessions

This Subclause modifies Subclause 4.43, "SQL-sessions", in ISO/IEC 9075-2.

4.16.1 SQL-session properties

This Subclause modifies Subclause 4.43.3, "SQL-session properties", in ISO/IEC 9075-2.

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.

— A pass-through flag.

— A pass-through foreign server name, if any.

— Zero or more {<statement name> : ExecutionHandle} pairs.

Insert this paragraph At the end of every SQL-session, every FSConnection handle that is contained in the SQL-session context is freed.

Insert this paragraph At the end of every SQL-session, every WrapperEnv handle that is contained in the SQL-session context is freed.

Insert this paragraph An SQL-session has a pass-through flag that is initially set to *False* when the SQL-session is started. The successful execution of a <set passthrough statement> that contains a <foreign server name> changes the pass-through flag to *True*. An SQL-session whose pass-through flag is *True* additionally has a pass-through foreign server name. Every time a <set passthrough statement> is executed, all {<SQL statement name> : ExecutionHandle} pairs are removed from the current SQL-session context. Every time a <set passthrough statement> that contains a <foreign server name> *FSN* is successfully executed, the pass-through foreign server name included the current SQL-session context is set to *FSN*. Every time a <prepare statement> is executed after a <set passthrough statement> that contains a <foreign server name> has been executed successfully, an {<SQL statement name> : ExecutionHandle} pair is made part of the current SQL-session context. Every time a <deallocate prepared statement> is successfully executed after a <set passthrough statement> that identifies a <foreign server name> has been executed, the corresponding {<SQL statement name> : ExecutionHandle} pair is removed from the current SQL-session context. Every time a <set passthrough statement> that specifies 'OFF' is executed, the pass-through flag is set to *False* and the pass-through foreign server name is deleted from the current SQL-session context.

4.17 Foreign-data wrapper interface

A foreign-data wrapper interface consists of the signatures of the routines that make up a given foreign-data wrapper. These routines serve the following purposes:

— Allocate and deallocate resources.

— Control connections to foreign servers.

— Receive data from the SQL-server about the foreign server request to be executed at the foreign server.

— Send data from the foreign server to the SQL-server about the foreign server request that the foreign server is willing to execute.

— Initiate and terminate the execution of foreign server requests by the foreign server.

4.17.1 Handles

A *handle* is a value of INTEGER data type that identifies an allocated resource that provides session state information about a foreign server, a foreign-data wrapper, or a foreign server session of interest to connected components of that session. Handles presented as arguments to invocations of foreign-data wrapper interface routines enable the invoker to give or obtain the information they reference. The handle of a particular resource is allocated by the keeper of the state information — either the foreign-data wrapper or the SQL-server — to enable the SQL-server or the foreign-data wrapper, respectively, to access that state 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, presented in approximately the order in which they are materialized in Table 2, “Sequence of actions during the execution of foreign server requests”. The operations that cause their creation and destruction are given in Table 2, “Sequence of actions during the execution of foreign server requests”.

— **WrapperEnv handle**: This handle is allocated by a foreign-data wrapper to reference information during the interaction with the SQL-server. It identifies an *allocated FDW-environment* and is allocated via a call from the SQL-server to the `AllocWrapperEnv()` routine. This handle shall be allocated before any foreign server requests are made to a foreign-data wrapper. It remains valid until the SQL-server invokes `FreeWrapperEnv(`*`WH`*`)`, where *WH* is the handle in question.

— **Server handle**: This handle is allocated by the SQL-server to reference a foreign server. A foreign-data wrapper uses this handle to obtain information about a foreign server to which 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 and presented by the SQL-server to a foreign-data wrapper by invoking `ConnectServer()`.

— **FSConnection handle**: This handle is allocated by a foreign-data wrapper to reference information about a foreign server session. It is allocated via a call to the `ConnectServer()` routine. This handle shall be allocated before any foreign server requests to be executed during that foreign server session are presented to a foreign-data wrapper.

— **Query Context handle**: This handle is allocated by the foreign-data wrapper to reference information that spans multiple foreign server requests. The SQL-server uses it to indicate to the foreign-data wrapper that identical value expression handles in the same query context (denoted by the same Query Context handle) but in different foreign server requests (represented by different Request handles) represent identical value expressions. It is foreign-data wrapper implementation-dependent whether the foreign-data wrapper uses this information to re-use the previously evaluated value expression or whether the foreign-data wrapper re-evaluates the value expression. The handle remains valid until the SQL-server invokes `FreeQueryContext()`.

— **Request handle**: This handle is allocated by the SQL-server to reference a foreign server request that is to be executed by a foreign server. A request handle may reference a simple foreign server request, such as `SELECT * FROM T`, or it may reference a complex foreign server request that includes predicates, joins, ordering, *etc*. A request handle is used by the foreign-data wrapper to retrieve (for example) the names of foreign tables referenced in the from clause, the names of column references in the select list, *etc.*, using foreign-data wrapper interface SQL-server routines. This handle is allocated implicitly.

— **Table Reference handle**: This handle is allocated by the SQL-server to reference 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 the SQL-server to reference a <value expression> contained in a foreign server request. This handle is allocated implicitly.

— **Reply handle**: This handle is allocated by a foreign-data wrapper to reference the subset of foreign server requests it is capable of executing. This handle is allocated via a call during a foreign server session to the `InitRequest()` routine and remains valid in that foreign server session until it is the argument to an invocation of `FreeReplyHandle()`.

— **Execution handle**: This handle is allocated by a foreign-data wrapper. In decomposition mode, it is used to reference the information the foreign-data wrapper needs to process the foreign server request referenced by the corresponding reply handle and the information associated with the data resulting from the processing of the foreign server request. This handle is allocated via a call to the `InitRequest()` routine, which sets the associated PASSTHROUGH flag to *False*. In pass-through mode, this handle is used to reference the information that the foreign-data wrapper needs to process the foreign server request that is sent to the foreign server. This handle is allocated via a call to the `TransmitRequest()` routine, which sets the associated PASSTHROUGH flag to *True*.

NOTE 15 — “decomposition mode” and “pass-through mode” are defined in Subclause 4.17.3.5, “Decomposition and pass-through modes”.

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

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

— **Descriptor handle**: This handle is allocated by either the SQL-server or a foreign-data wrapper to reference a foreign-data wrapper descriptor area.

— **Routine Mapping handle**: This handle is implicitly allocated by the SQL-server to reference an allocated routine mapping description.

4.17.2 Foreign server sessions

A *foreign server session* is the sequence of operations performed by a foreign-data wrapper on a particular FSConnection handle during the existence of that handle.

A foreign server session on FSConnection handle *FSCH* begins with the invocation of `ConnectServer()` that brings *FSCH* into existence and ends with the invocation of `FreeFSConnection(`*`FSCH`*`)`.

4.17.3 Foreign-data wrapper interface routines

The terms *foreign-data wrapper interface SQL-server routine* and *foreign-data wrapper interface wrapper routine* are used to distinguish routines provided by the SQL-server from routines provided by a foreign-data wrapper, respectively. A foreign-data wrapper interface routine that is both an SQL-server routine and a wrapper routine is referred to as a *foreign-data wrapper interface general routine*.

The foreign-data wrapper interface routines of a given implementation are either all functions or all procedures, the choice being implementation-defined. They are functions if their return codes are values returned by their invocations and they are procedures if their return codes are instead assigned to an output parameter named ReturnCode. The specific terms *foreign-data wrapper interface function* and *foreign-data wrapper interface procedure* are used when it is necessary to distinguish between the two kinds.

4.17.3.1 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.

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

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

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

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

— **GetTableRefElem**: 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>.

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

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

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

— **GetSelectElem**: 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>.

— **GetSelectElemType**: This routine returns the kind of a <value expression> given the value expression handle. Possible return values are COLUMN_NAME, OPERATOR, PARAMETER, and CONSTANT.

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

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

— **GetReplyTableRef**: 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 `GetNumReplyTableRefs()` routine.

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

— **GetReplySelectElem**: This routine returns the number of the select list element in the original foreign server 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 `GetNumReplySelectElems()` routine.

— **GetNumReplyBoolVE**: This routine returns the number of <boolean value expression>s simply contained in the <where clause> of the original foreign server request that the foreign-data wrapper is capable of handling, given a reply handle.

— **GetReplyBoolVE**: This routine returns the number of a <boolean value expression> element from the <where clause> in the original foreign server request that the foreign-data wrapper is capable of handling, given a reply handle and a number that ranges from 1 (one) to the value returned by the `GetNumReplyBoolVE()` routine.

— **GetReplyDistinct**: This routine returns information identifying whether the foreign-data wrapper is capable of providing distinct rows in the result set, given a reply handle.

— **GetReplyCardinality**: This routine returns an estimate of the cardinality of the result set associated with the reply, given a reply handle.

— **GetReplyFirstCost**: This routine returns a value that represents the estimated cost to retrieve the first row of the result set associated with the reply, given a reply handle. Larger values represent greater costs.

— **GetReplyExecCost**: This routine returns a value that represents the estimated cost to retrieve the result set associated with the reply, given a reply handle. Larger values represent greater costs.

— **GetReplyReExecCost**: This routine returns a value that represents the estimated cost to re-execute the reply, given a reply handle. Larger values represent greater costs.

— **GetNumReplyOrderBy**: This routine returns the number of columns that are used to order the result that the foreign-data wrapper is capable of handling, given a reply handle.

— **GetReplyOrderElem**: This routine returns the number of a <value expression> from the <select list> used to order the result in the original foreign server request that the foreign-data wrapper is capable of handling, given a reply handle and a number that ranges from 1 (one) to the value returned by the `GetNumReplyOrderBy()` routine.

— **GetNextReply**: This routine returns a new reply handle and execution handle for the original foreign server request, given a reply handle.

— **GetNumBoolVE**: This routine returns the number of <boolean value expression>s simply contained in the <where clause> of a <query specification>, given a request handle.

— **GetBoolVE**: This routine returns a handle for a <boolean value expression> from the <where clause> of a <query specification>, given a request handle and a number that ranges from 1 (one) to the value returned by the `GetNumBoolVE()` routine.

— **GetDistinct**: This routine returns information whether the query specifies DISTINCT or ALL, given a request handle.

— **GetNumOrderByElems**: This routine returns the number of columns that are used to order the result, given a request handle.

— **GetOrderByElem**: This routine returns a handle for a <value expression> used to order the result of a query, given a request handle and a number that ranges from 1 (one) to the value returned by the `GetNumOrderByElems()` routine.

— **GetValueExpKind**: This routine returns the kind of a <value expression>, given a value expression handle. Possible return values are COLUMN_NAME, OPERATOR, PARAMETER, CONSTANT.

— **GetNumChildren**: This routine returns the number of <value expression>s immediately contained in the containing <value expression>, given a value expression handle.

— **GetVEChild**: This routine returns a handle for a <value expression> immediately contained in the containing <value expression>, given a value expression handle and a number that ranges from 1 (one) to the value returned by the `GetNumChildren()` routine.

— **GetValueExpName**: This routine returns the name associated with a <value expression>, given a value expression handle.

— **GetValueExpTable**: This routine returns a table reference handle with which the table associated with the <value expression> is associated.

— **GetValueExpDesc**: This routine returns a handle for a value expression descriptor describing a <value expression>.

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

— **GetTableColOpt**: 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.

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

— **GetTableOpt**: 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.

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

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

— **GetNumTableColOpts**: 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.

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

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

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

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

— **GetUserOptByName**: 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.

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

— **GetWrapperOptByName**: 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.

— **GetSQLString**: This routine returns a character string representation of the query that is associated with the request handle.

— **GetRoutineMapping**: This routine returns the routine mapping handle for an allocated routine mapping description, given a value expression handle.

— **GetRoutMapOptName**: This routine returns the generic option value, given the routine mapping handle and the name of the option.

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

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

4.17.3.2 Initialization routines

— **AdvanceInitRequest**: This routine is used by the SQL-server to cause a foreign server request to be prepared. The routine has three input parameters: a previously allocated FSConnection handle, a previously allocated QueryContext handle, and a request handle that describes the foreign server request. The routine has two output parameters: a reply handle that describes how much of the foreign server 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 foreign server request and the data rows that will be returned. This routine uses the `InitRequest()` routine multiple times to generate multiple reply handle/execution handle pairs. Additional reply handle/execution handle pairs can be retrieved by the SQL-server using the `GetNextReply()` routine.

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

— **AllocQueryContext**: This routine is used by the SQL-server to retrieve a Query Context handle that the SQL-server will use to indicate to the foreign-data wrapper that identical Value Expression handles in different foreign server requests refer to identical value expressions.

— **AllocWrapperEnv**: This routine is used by the SQL-server to allow the 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.

— **ConnectServer**: 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. ConnectServer 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.

— **GetOpts**: 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. This routine is

invoked by the SQL-server, and executed by the foreign-data wrapper, whenever the SQL-server requires information about options supported by the foreign-data wrapper and the foreign server. It is thus invoked under implementation-dependent circumstances.

— **InitRequest**: This routine is used to generate a reply handle and an execution handle for a given foreign server request. The routine has two input parameters: a previously allocated FSConnection handle, and a request handle that describes the foreign server request. The routine has two output parameters: a reply handle that describes how much of the foreign server 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 foreign server request and the data rows that will be returned.

4.17.3.3 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 an invocation of either `AdvanceInitRequest()` or `GetNextReply()`). 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 row with the ExecutionHandle. The SQL-server may invoke this routine until all data is returned.

— **ReOpen**: This routine may be 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 returned by either the `AdvanceInitRequest()` routine or the `GetNextReply()` routine). This routine allows an 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 a foreign server request that initiated the communication with the foreign-data wrapper. The routine has one input parameter: a previously allocated ExecutionHandle.

— **GetStatistics**: This routine is used by the SQL-server to request statistics, if any, related to the foreign server request previously sent to the foreign-data wrapper. Such statistics are entirely implementation-defined in nature. This routine is invoked by the SQL-server, and executed by the foreign-data wrapper, whenever the SQL-server requires statistics that may be provided by the foreign-data wrapper and the foreign server. It is thus invoked under implementation-dependent circumstances.

— **TransmitRequest**: This routine is used by the SQL-server to transmit a foreign server request in the native language of the foreign server to the foreign server in pass-through mode. The foreign server analyzes the transmitted foreign server request and returns information about that foreign server request to the SQL-server. This information includes: Whether the foreign server request requires one or more input values in order to be executed; and whether the foreign server request returns one or more result values upon execution. This information is associated with the descriptors attached to the execution handle that is the output parameter of this routine. This routine has three input parameters: a previously allocated FSConnection handle, a string containing the foreign server request, and an integer indicating the string length.

4.17.3.4 Termination routines

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

— **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 parameter: a previously allocated FSConnection handle.

— **FreeQueryContext**: This routine is used by the SQL-server to allow the foreign-data wrapper to free all resources it may have associated with a Query Context handle. This routine has one input parameter, a previously allocated Query Context handle.

— **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, a previously allocated ReplyHandle.

— **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, a previously allocated WrapperEnv handle.

4.17.3.5 Decomposition and pass-through modes

Depending on whether the pass-through flag in the current SQL-session context is set to *True* or *False*, the SQL-server is said to be either in *pass-through mode* or *decomposition mode*. When the SQL-server is in decomposition mode, the SQL-server analyzes the SQL-client request and invokes the `AdvanceInitRequest()` routine to communicate foreign server request to the foreign-data wrapper. When the SQL-server is in pass-through mode, the SQL-server does not analyze the SQL-client request and invokes the `TransmitRequest()` routine to communicate foreign server request to the foreign-data wrapper.

4.17.3.6 Sequence of actions during the execution of foreign server requests

For decomposition mode, Table 2, “Sequence of actions during the execution of foreign server requests”, shows the sequence of actions as described by the General Rules of Subclause 7.1, “<table reference>”, when a foreign table is identified by the <table name> simply contained in the <table reference>.

For pass-through mode, Table 2, “Sequence of actions during the execution of foreign server requests”, shows the sequence of actions that is likely to occur when an SQL-client requests the preparation and execution of statements using dynamic SQL.

Table 2 — Sequence of actions during the execution of foreign server requests

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
1	Receives a query from SQL-client that involves data from a foreign table in a <table reference>. Determines relationship of foreign table → foreign server → foreign-data wrapper.	Receives from the SQL-client a statement to be executed in pass-through mode that involves one foreign server. Determines relationship of the foreign server to the foreign-data wrapper.			
2	Creates a **WrapperHandle** and associates information about the foreign-data wrapper with that handle.				
3	Invokes the **AllocWrapperEnv (WrapperHandle, WrapperEnvHandle)** routine to initialize the foreign data wrapper.		⇒		
4			⇐	Invokes the **Get... (WrapperHandle, ...)** routines in the SQL-server to retrieve information about the foreign-data wrapper that the SQL-server has stored in its Information Schema.	
NOTE 16 — This information is provided in the <foreign-data wrapper definition>.					
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**.				
NOTE 17 — If the current SQL-session context already includes a {foreign-data wrapper name : WrapperEnvHandle} pair that could be used, then Step **2** through Step **7** are optional.					
8	Creates a **ServerHandle** and associates information about the foreign server with that handle.				

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
9	Creates a **UserHandle** and associates information about the current user with that handle.				
10	Invokes the **ConnectServer (WrapperEnvHandle, ServerHandle, UserHandle, FSConnectionHandle)** routine 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 that the SQL-server has stored in its Information Schema.	
NOTE 18 — This information is provided in the <foreign server definition>.					
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 that the SQL-server has stored in its Information Schema.	
NOTE 19 — This information is provided in the <user mapping definition>.					
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**.				
NOTE 20 — If the current SQL-session context already includes a {foreign server name : FSConnectionHandle} pair that could be used, then Step **8** Step **17** are optional.					

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
18	Invokes the **AllocQueryContext()** routine in the foreign-data wrapper to obtain a **QueryContextHandle**.		⇒		
19				Allocates data structures to utilize the information about the query context and associates them with the **QueryContextHandle**.	
20	Creates a **RequestHandle** and associates with it the information about the part of query that could be handled by the foreign-data wrapper.				
21	Creates as many **TableReferenceHandle**s as are needed and associates with each of them the information about a particular <table reference>.				
22	Creates a **TableReferenceDescriptor (TRD)** for the result of the <query specification> described by the **RequestHandle**, and sets all the fields with details about each of the columns.				

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
23	Creates a **WrapperParameterDescriptor (WPD)** to describe the <dynamic parameter specification>s in the <query specification> described by the **RequestHandle**, abd sets all the fields with details about each of the <dynamic parameter specification>s.				
24	Creates as many **ValueExpressionHandle**s as are needed and associates with each of them the information about a particular <value expression> in the <select list> and <where clause>, respectively.				
25	Invokes the **AdvanceInitRequest (FSConnectionHandle, RequestHandle, ReplyHandle, ExecutionHandle, QueryContextHandle)** routine in the foreign-data wrapper to find out how much of the request the foreign server can actually process.	Invokes the **TransmitRequest (FSConnectionHandle, RequestString, StringLength, ExecutionHandle)** routine to allow the foreign-data wrapper and the foreign server to analyze the foreign server request.	⇒		

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
26			⇐	Invokes the **Get... (RequestHandle, ...)**, **GetTRDHandle (TableReference-Handle)**, **GetDescriptor (TRD)**, **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.	Executes the **TransmitRequest ()** routine.
NOTE 21 — The foreign-data wrapper could invoke the **GetSQLString (RequestHandle, StringFormat, SQLString, BufferLength, StringLength)** routine in the SQL-server to examine the foreign server request and to determine how much of the foreign server request the foreign-data wrapper can handle, instead of the **Get... (RequestHandle, ...)**, **Get... (TableReferenceHandle, ...)**, **Get... (ValueExpressionHandle, ...)**, **GetTRDHandle (TableReferenceHandle)**, and **GetDescriptor (TRD)** routines.					
27	Executes the **Get... (RequestHandle, ...)**, **GetTRDHandle (TableReference-Handle)**, **GetDescriptor (TRD)**, **Get... (TableReferenceHandle, ...)**, and **Get... (ValueExpressionHandle, ...)** routines as requested by the foreign-data wrapper.		⇒		
NOTE 22 — If the foreign-data wrapper invoked the **GetSQLString (RequestHandle, StringFormat, SQLString, BufferLength, StringLength)** routine, then this routine is executed instead of the **Get... (RequestHandle, ...)**, **Get... (TableReferenceHandle, ...)**, **Get... (ValueExpressionHandle, ...)**, **GetTRDHandle (TableReferenceHandle)**, and **GetDescriptor (TRD)** routines.					

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
28				Creates a **ReplyHandle** and associates with it the information about the part of query that could actually be handled by the foreign-data wrapper.	
29			⇐	Creates an **ExecutionHandle** and associates with it the information about the actual execution plan.	
30			⇐	Invokes the **AllocDescriptor ()** routine to create two descriptors (SRD and SPD) and associates both of them with the **ExecutionHandle**. Initializes the descriptors with default values wherever applicable.	Invokes the **AllocDescriptor ()** routine to create two descriptors (WRD and SRD) to describe the columns of the result table and associates both of them with the **ExecutionHandle**. Initializes both the descriptors with default values wherever applicable. Sets the fields in the WRD to correspond to the result columns associated with the **ExecutionHandle**.
31	Executes the **AllocDescriptor ()** routine as requested by the foreign-data wrapper.		⇒		

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
32			⇐		Invokes the **AllocDescriptor ()** routine to create two descriptors, WPD and SPD, to describe <dynamic parameter specification>s. Associates both of them with the **ExecutionHandle**. Initializes both the descriptors with default values wherever applicable. Sets the fields in the WPD to correspond to the dynamic parameters associated with the **ExecutionHandle**.
33		Executes the **AllocDescriptor ()** routine as requested by the foreign-data wrapper.	⇒		
34				Repeats steps Step **25** through Step **33** to create multiple **ReplyHandle** / **ExecutionHandle** pairs for the same foreign server request.	
NOTE 23 — Step **34** is optional.					
35	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.		⇒		

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
36			⇐	Executes the **Get...(ReplyHandle, ...)** routines as requested by the SQL-server.	
37	Invokes the **GetNextReply( )** routine to retrieve another **ReplyHandle** for the same foreign server request		⇒		
38			⇐	Executes the **GetNextReply()** routine and returns a **ReplyHandle** and **ExecutionHandle**.	
39	Repeats Step **35** through Step **38**.				
NOTE 24 — Step **37** through Step **38** 39 are optional.					
40	Invokes the **FreeReplyHandle (ReplyHandle)** routine in the foreign-data wrapper to indicate that the **ReplyHandle** is no longer required.		⇒		
41				Frees resources associated with **ReplyHandle**.	
42	Repeats steps Step **20** through Step **41**.				
NOTE 25 — Step **42** is optional.					

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
43	Invokes the **GetSRDHandle (ExecutionHandle)** routine to get the SRD.	Invokes the **GetWRDHandle (ExecutionHandle)** and **GetSRDHandle (ExecutionHandle)** routines to get the WRD and the SRD, respectively.	⇒		
44			⇐	Executes the **GetSRDHandle (ExecutionHandle)** routine as requested by the SQL-server.	Executes the **GetWRDHandle (ExecutionHandle)** and **GetSRDHandle (ExecutionHandle)** routines as requested by the SQL-server.
45		Invokes the **GetDescriptor (WRD)** routine multiple times to get all the information associated with WRD.			
46	Invokes the **SetDescriptor (SRD)** routine multiple times to populate appropriate fields in SRD.				
NOTE 26 — In pass-through mode, **SetDescriptor (SRD)** will only be invoked if results are returned by the foreign-data server.					
47	Invokes the **GetSPDHandle (ExecutionHandle)** routine to get the SPD.	Invokes the **GetWPDHandle (ExecutionHandle)** and **GetSPDHandle (ExecutionHandle)** routines to get the WPD and SPD, respectively.	⇒		
48			⇒	Executes the **GetSPDHandle (ExecutionHandle)** routine as requested by the SQL-server.	Executes the **GetWPDHandle (ExecutionHandle)** and **GetSPDHandle (ExecutionHandle)** routines as requested by the SQL-server.

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
49		Invokes **GetDescriptor (WPD)** multiple times to obtain the information associated with WPD.			
50	If there are any dynamic parameters present, invokes **SetDescriptor (SPD)** multiple times to populate appropriate fields in SPD.		⇒		
51	Frees the **RequestHandle**.				
52	Frees each of the **TableReferenceHandle**s.				
53	Frees each of the **ValueExpressionHandle**s.				
54	Invokes the **Open (ExecutionHandle)** routine in the foreign-data wrapper to initiate the execution of the foreign server request in the foreign-data wrapper.		⇒		
55			⇐	Executes the **Open (ExecutionHandle)** routine as requested by the SQL-server.	
56	Invokes the **Iterate (ExecutionHandle)** routine in the foreign-data wrapper to retrieve a row.		⇒		
57			⇐	Performs the work needed to retrieve the next row from the foreign server and associates it with the **ExecutionHandle**.	
58	Repeats Step **56** through Step **57** until all data is retrieved.		⇔		

NOTE 27 — In pass-through mode, Step **56** through Step **58** steps 56 through 58 are executed only if the foreign-data wrapper returns a set of rows.

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
59	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.		⇒		
60				Optional: Executes the **ReOpen (ExecutionHandle)** routine as requested by the SQL-server to 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*.	
61	Completes the work necessary to answer the SQL-client's query. As a result, invokes **Close (ExecutionHandle)** routine in the foreign-data wrapper.	Completes the work necessary to answer the SQL-client's statement in pass-through mode. Possibly invokes the **Close (ExecutionHandle)** routine in the foreign-data wrapper.	⇒		
62				Executes the **Close (ExecutionHandle)** routine as requested by the SQL-server.	

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	**Pass-through**		**Decomposition**	**Pass-through**
63	Invokes **FreeExecutionHandle (ExecutionHandle)** routine in the foreign-data wrapper.		⇒		
64				Frees resources associated with **ExecutionHandle**.	
65			⇐	Invokes the **FreeDescriptor ()** routine with the **SRDHandle** as the input argument.	Invokes the **FreeDescriptor ()** routine four times with the **SRDHandle**, **SPDHandle**, **WRDHandle**, and **WPDHandle**, respectively, as the input arguments.
66	Executes the **FreeDescriptor ()** routine as requested by the foreign-data wrapper.				
67	Invokes the **FreeFSConnection (FSConnectionHandle)** routine in the foreign-data wrapper.		⇒		
68				Frees resources associated with **FSConnectionHandle**.	
NOTE 28 — Step **67** and Step **68** are optional, if the SQL-server wants to reuse the FSConnectionHandle.					
69	Invokes **FreeWrapperEnv (WrapperEnvHandle)** routine in the foreign-data wrapper.		⇒		
70				Frees resources associated with **WrapperEnvHandle**.	
NOTE 29 — Step **69** and Step **70**\ are optional, if the SQL-server wants to reuse the WrapperEnvHandle.					

4.17.4 Return codes

The execution of a foreign-data wrapper 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 invoking a foreign-data wrapper interface function or as the value of the ReturnCode argument resulting from invoking a foreign-data wrapper interface 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.

NOTE 30 — Foreign-data wrapper functions and foreign-data wrapper procedures are defined in Subclause 22.1, "<foreign-data wrapper interface routine>".

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

— A value of 1 (one) indicates **Success with information**. The foreign-data wrapper interface routine executed successfully but a completion condition was raised: *warning*.

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

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

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

If the foreign-data wrapper interface 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 foreign-data wrapper interface routines other than `GetDiagnostics()`, the implementation records information about completion conditions and about exception conditions raised other than *FDW-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.

4.17.5 Foreign-data wrapper diagnostics areas

Each diagnostics area consists of 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 foreign-data wrapper interface 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 foreign-data wrapper interface 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: *FDW-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 code.

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 SQL-server or foreign-data wrapper 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 does not modify the identified diagnostics area.

A foreign-data wrapper diagnostics area consists of the fields specified in Table 3, "Fields used in foreign-data wrapper diagnostics areas".

Table 3 — Fields used in foreign-data wrapper diagnostics areas

Field	Data type
MORE	INTEGER
NUMBER	INTEGER
RETURNCODE	SMALLINT
Implementation-defined header field	Implementation-defined
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 foreign-data wrapper interface routines.

4.17.6 Null pointers

If the 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: *FDW-specific condition — invalid use of null pointer*.

4.17.7 Foreign-data wrapper descriptor areas

A foreign-data wrapper descriptor area provides an interface for a description of values required for the execution of a foreign server request in either decomposition mode or in pass-through mode by a foreign-data wrapper and for a description of values resulting from such an execution.

Each foreign-data wrapper descriptor area comprises header fields and zero or more foreign-data wrapper item descriptor areas. The header and item descriptor area fields are specified in Table 4, "Fields in foreign-data wrapper descriptor areas". The header fields include a COUNT field that indicates the number of item descriptor areas.

Some host languages are able to access host variables whose addresses are stored in an item descriptor field named DATA_POINTER in a foreign-data wrapper descriptor area. Such languages are called *pointer-supporting languages* and include Ada, C, Pascal, and PL/I. Languages that cannot access variables whose addresses are stored in the DATA_POINTER field of a foreign-data wrapper descriptor are called *non-pointer-supporting languages*. Such languages include COBOL, Fortran, and M.

The `GetDescriptor()` routine enables information to be retrieved from any foreign-data wrapper descriptor area. The `SetDescriptor()` routine enables information to be set in any foreign-data wrapper descriptor area.

The following foreign-data wrapper descriptor areas are either implicitly or explicitly allocated and deallocated:

— Table Reference Descriptor (TRD): This descriptor is allocated automatically by the SQL-server to describe a foreign table referenced in a foreign server request, and is associated with a TableReferenceHandle created by the 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 by the SQL-server if a foreign-data wrapper requests its allocation. It is used to describe the result of a foreign server request to be executed by that foreign-data wrapper in pass-through mode, and is associated with an ExecutionHandle. The foreign-data wrapper uses the `SetDescriptor()` routine to set information in the WRD. 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 by the SQL-server if a foreign-data wrapper requests its allocation. It 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 by the SQL-server if a foreign-data wrapper requests its allocation. It is used to describe the input values required for the execution of a foreign server request by that foreign-data wrapper, and is associated with an ExecutionHandle. The foreign-data wrapper uses the `SetDescriptor()` routine to set information in the WPD. The SQL-server can obtain the handle to a WPD by invoking the `GetWPDHandle()` routine. It can then retrieve the information in that WPD by invoking the `GetDescriptor()` routine.

— Server Parameter Descriptor (SPD): This descriptor is allocated by the SQL-server if a foreign-data wrapper requests its allocation. It 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.

— Value expression descriptor: This descriptor is implicitly allocated by the SQL-server for each value expression contained in a foreign server request. It is used to describe the most specific type and value of each value expression in the foreign server request. The foreign-data wrapper can obtain a handle to this descriptor by invoking the `GetValueExpDesc()` routine. It can retrieve information in that descriptor by invoking the `GetDescriptor()` routine.

Table 4 — Fields in foreign-data wrapper descriptor areas

Field	Data Type
COUNT	SMALLINT
DYNAMIC_FUNCTION	CHARACTER VARYING(*L*[1])
DYNAMIC_FUNCTION_CODE	INTEGER
KEY_TYPE	SMALLINT
TOP_LEVEL_COUNT	SMALLINT
Implementation-defined foreign-data wrapper descriptor header field	Implementation-defined
CARDINALITY	INTEGER
CHARACTER_SET_CATALOG	CHARACTER VARYING(*L*[1])

Field	Data Type
CHARACTER_SET_NAME	CHARACTER VARYING(*L*[1])
CHARACTER_SET_SCHEMA	CHARACTER VARYING(*L*[1])
COLLATION_CATALOG	CHARACTER VARYING(*L*[1])
COLLATION_NAME	CHARACTER VARYING(*L*[1])
COLLATION_SCHEMA	CHARACTER VARYING(*L*[1])
CURRENT_TRANSFORM_GROUP	CHARACTER VARYING(*L1*[1])
DATA	ANY
DATA_POINTER	host variable address
DATETIME_INTERVAL_CODE	SMALLINT
DATETIME_INTERVAL_PRECI-SION	SMALLINT
DEGREE	INTEGER
INDICATOR	INTEGER
KEY_MEMBER	SMALLINT
LENGTH	INTEGER
LEVEL	INTEGER
NAME	CHARACTER VARYING(*L*[1])
NULLABLE	SMALLINT
OCTET_LENGTH	INTEGER
PARAMETER_MODE	SMALLINT
PARAMETER_ORDINAL_POSITION	SMALLINT
PARAMETER_SPECIFIC_CATALOG	CHARACTER VARYING(*L*[1])
PARAMETER_SPECIFIC_NAME	CHARACTER VARYING(*L*[1])
PARAMETER_SPECIFIC_SCHEMA	CHARACTER VARYING(*L*[1])

Field	Data Type
PRECISION	SMALLINT
RETURNED_CARDINALITY	INTEGER
RETURNED_OCTET_LENGTH	INTEGER
SCALE	SMALLINT
SCOPE_CATALOG	CHARACTER VARYING(*L*[1])
SCOPE_NAME	CHARACTER VARYING(*L*[1])
SCOPE_SCHEMA	CHARACTER VARYING(*L*[1])
SPECIFIC_TYPE_CATALOG	CHARACTER VARYING(*L*[1])
SPECIFIC_TYPE_NAME	CHARACTER VARYING(*L*[1])
SPECIFIC_TYPE_SCHEMA	CHARACTER VARYING(*L*[1])
TYPE	SMALLINT
UNNAMED	SMALLINT
USER_DEFINED_TYPE_CATALOG	CHARACTER VARYING(*L*[1])
USER_DEFINED_TYPE_NAME	CHARACTER VARYING(*L*[1])
USER_DEFINED_TYPE_SCHEMA	CHARACTER VARYING(*L*[1])
Implementation-defined foreign-data wrapper descriptor item field	Implementation-defined

[1] **Where** *L* is an implementation-defined integer not less than 128, and *L1* is the implementation-defined maximum length for the <general value specification> CURRENT_TRANSFORM_GROUP_FOR_TYPE.

4.18 Introduction to SQL/CLI

This Subclause modifies Subclause 4.1, "Introduction to SQL/CLI", in ISO/IEC 9075-3.

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.

(Blank page)

5 Lexical elements

This Clause modifies Clause 5, "Lexical elements", in ISO/IEC 9075-2.

5.1 <token> and <separator>

This Subclause modifies Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

Function

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

Format

```
<non-reserved word> ::=

    !! All alternatives from ISO/IEC 9075-2
  | BLOCKED

  | CONTROL

  | DB

  | FILE | FS

  | INTEGRITY

  | LIBRARY | LIMIT | LINK

  | MAPPING

  | OFF

  | PASSTHROUGH | PERMISSION

  | RECOVERY | REQUIRING | RESTORE

  | SELECTIVE | SERVER

  | TOKEN

  | UNLINK

  | VERSION

  | WRAPPER

  | YES
```

```
<reserved word> ::=
    !! All alternatives from ISO/IEC 9075-2
  | DATALINK | DLNEWCOPY | DLPREVIOUSCOPY | DLURLCOMPLETE | DLURLCOMPLETEWRITE
  | DLURLCOMPLETEONLY | DLURLPATH | DLURLPATHWRITE | DLURLPATHONLY
  | DLURLSCHEME | DLURLSERVER | DLVALUE

  | IMPORT
```

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 Names and identifiers

This Subclause modifies Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2.

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>

<option name> ::=
  <identifier body>

<routine mapping name> ::=
  <identifier>
```

Syntax Rules

1) Insert this SR If a <foreign server name> does not contain a <catalog name>, then

   Case:

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

   b) Otherwise, 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

   Case:

   a) If the <foreign-data wrapper name> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default catalog name for the SQL-session is implicit.

   b) Otherwise, the <catalog name> that is specified or implicit for the SQL-client module is implicit.

3) Insert this SR In an <option name>, the number of <identifier part>s shall be less than 128.

4) Insert this SR The case-normal form of the <identifier body> of an <option name> is used for purposes such as and including determination of option name equivalence, representation in the Definition and Information Schemas, and representation in the diagnostics areas.

5) Insert this SR Two <option name>s are equivalent if the case-normal forms of their <identifier body>s, considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_IDENTIFIER and a collation *IDC* that is sensitive to case, compare equal according to the comparison rules in Subclause 8.2, "<comparison predicate>", in [ISO9075-2].

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.

3) Insert this GR A <routine mapping name> identifies a routine mapping.

Conformance Rules

No additional Conformance Rules.

6 Scalar expressions

This Clause modifies Clause 6, "Scalar expressions", in ISO/IEC 9075-2.

6.1 <data type>

This Subclause modifies Subclause 6.1, "<data type>", in ISO/IEC 9075-2.

Function

Specify a data type.

Format

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

<datalink type> ::=
  DATALINK [ <datalink control definition> ]

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

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

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

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

<write permission option> ::=
    WRITE PERMISSION FS
  | WRITE PERMISSION ADMIN <access token indication>
  | WRITE PERMISSION BLOCKED

<access token indication> ::=
    REQUIRING TOKEN FOR UPDATE
  | NOT REQUIRING TOKEN FOR UPDATE

<recovery option> ::=
    RECOVERY NO
```

```
  | RECOVERY YES

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

Syntax Rules

1) Insert this SR DATALINK specifies the datalink type.

2) Insert this SR If <data type> specifies DATALINK and <datalink control definition> is not specified, then NO LINK CONTROL is implicit.

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 either WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN shall be specified.

   c) If either WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN 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 either WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN shall be specified.

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

   NOTE 31 — Valid combinations of <datalink file control option> resulting from this Syntax Rule are shown in Table 1, "Valid datalink file control options".

4) Insert this SR If <datalink control definition> is specified, then <data type> shall not be contained in an <SQL variable declaration>.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR If <data type> is a <datalink type>, then a datalink type descriptor *DTD* is created. The link control options of *DTD* are:

   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, WRITE PERMISSION ADMIN, or WRITE PERMISSION BLOCKED is specified. If WRITE PERMISSION ADMIN is specified, then additionally the access token indication, according to whether REQUIRING TOKEN FOR UPDATE or NOT REQUIRING TOKEN FOR UPDATE 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

1) Without Feature M001, “Datalinks”, conforming SQL language shall not contain a <datalink type>.

6.2 <cast specification>

This Subclause modifies Subclause 6.13, "<cast specification>", in ISO/IEC 9075-2.

Function

Specify a data conversion.

Format

No additional Format items.

Syntax Rules

1) Insert before SR 2) *TD* shall not contain a <datalink control definition>.

2) Insert after SR 7) Add a new rightmost column to the table in the 7th paragraph:

SD	*TD*
	DL
EN	N
AN	N
C	N
D	N
T	N
TS	N
YM	N
DT	N
BO	N
UDT	N
B	N
RT	N
CT	N
RW	N

3) Insert after SR 7) Add new rows at the end of the table in the 7th paragraph:

DL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y

Where:

DL = Datalink

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 21) If *TD* and *SD* are datalink types, then *TV* is *SV*.

Conformance Rules

No additional Conformance Rules.

6.3 <value expression>

This Subclause modifies Subclause 6.28, "<value expression>", in ISO/IEC 9075-2.

Function

Specify a value.

Format

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

Syntax Rules

1) Replace SR 2) The declared type of a <common value expression> is the declared type of the <numeric value expression>, <string value expression>, <datetime value expression>, <interval value expression>, <user-defined type 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.

6.4 <string value function>

This Subclause modifies Subclause 6.32, "<string value function>", in ISO/IEC 9075-2.

Function

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

Format

```
<string value function> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <url complete expression>
  | <url complete for write expression>
  | <url complete only expression>
  | <url path expression>
  | <url path for write expression>
  | <url path only expression>
  | <url scheme expression>
  | <url server expression>

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

<url complete for write expression> ::=
  DLURLCOMPLETEWRITE <left paren> <datalink value expression> <right paren>

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

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

<url path for write expression> ::=
  DLURLPATHWRITE <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) Replace SR 1) The declared type of <string value function> is the declared type of the immediately contained <character value function>, <binary value function>, <JSON value constructor>, <JSON query>, <url complete expression>, <url complete for write expression>, <url complete only expression>, <url path expression>, <url path for write expression>, <url path only expression>, <url scheme expression>, or <url server expression>.

2) Insert this SR Let *DLCS* be the <character set name> of the datalink character set.

NOTE 32 — "datalink character set" is defined in Subclause 4.8, "Datalinks".

3) Insert this SR Let *DVE* be the <datalink value expression>.

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

5) Insert this SR The declared type of <url complete for write expression> is variable-length character string with character set *DLCS* and an implementation-defined maximum length.

6) Insert this SR The declared type of <url complete only expression> is variable-length character string with character set *DLCS* and an implementation-defined maximum length.

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

8) Insert this SR The declared type of <url path for write expression> is variable-length character string with character set *DLCS* and an implementation-defined maximum length.

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

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

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

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) The result of <string value function> is the result of the immediately contained <character value function>, <binary value function>, <JSON value constructor>, <JSON query>, <url complete expression>, <url complete for write expression>, <url complete only expression>, <url path expression>, <url path for write expression>, <url path only expression>, <url scheme expression>, or <url server expression>.

2) 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.

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

Case:

a) If *DV* is SQL-mediated, then the result is the File Reference of *DV* combined with a read token in an implementation-dependent manner.

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

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

Case:

a) If *DV* is SQL-mediated and the SQL-Mediated Write Access Indication of *DV* is *True*, then the result is the File Reference of *DV* combined with a write token in an implementation-dependent manner.

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

5) Insert this GR If <url complete only expression> is specified, then the result is the File Reference of *DV*.

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

Case:

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

Case:

i) If *DV* is SQL-mediated, then *HP* combined with a read token in an implementation-dependent manner.

ii) Otherwise, *HP*.

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

Case:

i) If *DV* is SQL-mediated, then *FP* combined with a read token in an implementation-dependent manner.

ii) Otherwise, *FP*.

c) If the File Reference of *DV* conforms to an implementation-defined format, then an implementation-defined value.

d) Otherwise, a zero-length character string.

7) Insert this GR If <url path for write expression> is specified, then the result is

Case:

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

Case:

i) If *DV* is SQL-mediated and the SQL-Mediated Write Access Indication of *DV* is *True*, then *HP* combined with a write token in an implementation-dependent manner.

ii) Otherwise, *HP*.

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

Case:

i) If *DV* is SQL-mediated and the SQL-Mediated Write Access Indication of *DV* is *True*, then *FP* combined with a write token in an implementation-dependent manner.

ii) Otherwise, *FP*.

c) If the File Reference of *DV* conforms to an implementation-defined format, then an implementation-defined value.

d) Otherwise, a zero-length character string.

8) Insert this GR If <url path only expression> is specified, then the result is

Case:

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

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

c) If the File Reference of *DV* conforms to an implementation-defined format, then an implementation-defined value.

d) Otherwise, a zero-length character string.

9) Insert this GR If <url scheme expression> is specified, then the result is

Case:

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

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

c) If the File Reference of *DV* conforms to an implementation-defined format, then an implementation-defined value.

d) Otherwise, a zero-length character string.

10) Insert this GR If <url server expression> is specified, then the result is

Case:

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

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

c) If the File Reference of *DV* conforms to an implementation-defined format, then an implementation-defined value.

d) Otherwise, a zero-length character string.

Conformance Rules

No additional Conformance Rules.

6.5 <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 contain a <datalink value expression>.

6.6 <datalink value function>

Function

Specify a function yielding a datalink value.

Format

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

<datalink value constructor> ::=
    DLVALUE <left paren> <data location> <right paren>
  | DLNEWCOPY <left paren> <data location> <comma> <token indication> <right paren>
  | DLPREVIOUSCOPY <left paren> <data location> <comma> <token indication> <right paren>

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

<token indication> ::=
  <unsigned integer>
```

Syntax Rules

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

2) The declared type of a <datalink value function> is the declared type of its <datalink value constructor>.

3) The character set name of the declared type of <data location> shall be equivalent to the character set name of the datalink character set.

NOTE 33 — "datalink character set" is defined in Subclause 4.8, "Datalinks".

Access Rules

None.

General Rules

1) Let *DLOC* be the result of evaluating <data location>.

   a) If DLVALUE is specified and *DLOC* is the null value, then the result of *DVC* is the null value.

   b) If either DLNEWCOPY or DLPREVIOUSCOPY is specified, then:

      i) Let *TIV* the result of evaluating <token indication>.

      ii) If *TIV* is neither equal to 0 (zero) nor 1 (one), then an exception condition is raised: *data exception — invalid parameter value*.

      iii) If *DLOC* is the null value, then an exception condition is raised: *data exception — null argument passed to datalink constructor*.

iv) If *TIV* is equal to 1 (one) and if the write token included in DLOC does not conform to implementation-defined requirements, then an exception condition is raised: *datalink exception — invalid write token*.

c) Case:

i) If DLVALUE is specified, then let *DLOCWOT* be *DLOC*.

ii) Otherwise:

1) If *TIV* is equal to 0 (zero), then let *DLOCWOT* be *DLOC*.

2) If *TIV* is equal to 1 (one), then let *DLOCWOT* be *DLOC* without the write token included in *DLOC* and let *WT* be the write token included in *DLOC*.

d) If *DLOCWOT* conforms neither to the Format of Subclause 8.1, "URL format", nor to an implementation-defined format, then an exception condition is raised: *data exception — invalid data specified for datalink*.

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 34 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

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

i) The File Reference of *DL* is *DLOCWOT*.

Case:

1) If *DLOCWOT* conforms to the Format of Subclause 8.1, "URL format", then

Case:

A) If *DLOCWOT* contains an <http url>, then the <http>, <host>, and <hpath> contained in the <http url> are the *scheme* of *DL*, the *host* of *DL*, and the *path* of *DL*, respectively.

B) If *DLOCWOT* contains a <file url>, then the <file>, <host>, and <fpath> contained in the <file url> are the *scheme* of *DL*, the *host* of *DL*, and the *path* of *DL*, respectively.

2) Otherwise, the *scheme* of *DL*, the *host* of *DL*, and the *path* of *DL* are implementation-defined.

ii) The SQL-Mediated Read Access Indication of *DL* is *False*.

iii) The SQL-Mediated Write Access Indication of *DL* is *False*.

iv) Case:

1) If either DLNEWCOPY or DLPREVIOUSCOPY is specified and *TIV* is equal to 1 (one), then the Write Token of *DL* is *WT*.

2) Otherwise, the Write Token of *DL* is the null value.

v) Case:

1) If DLNEWCOPY is specified, then the Construction Indication of *DL* is NEWCOPY.

2) If DLPREVIOUSCOPY is specified, then the Construction Indication of *DL* is PREVIOUSCOPY.

3) Otherwise, the Construction Indication of *DL* is the null value.

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 contain a <datalink value function>.

7 Query expressions

This Clause modifies Clause 7, "Query expressions", in ISO/IEC 9075-2.

7.1 <table reference>

This Subclause modifies Subclause 7.6, "<table reference>", in ISO/IEC 9075-2.

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 GR 2)c). Otherwise, let *T* be the table specified by the <table name> simply contained in *TP*.

Case:

a) If *T* is a foreign table with <table name> *FTN*, then the result of *TP* 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 invocation of `AllocWrapperEnv()` 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 FSConnection-Handle 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) If there is a user mapping identified by the current authorization identifier, then let *UH* be the UserHandle allocated for that user mapping; otherwise, let *UH* be the UserHandle allocated for the user mapping identified by PUBLIC. The resource identified by *UH* is referred to as an *allocated user mapping description*.

C) Let *FSCH* be the FSConnectionHandle returned by invocation of the `Connect-Server()` 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 *QCH* be the QueryContextHandle returned by invocation of the `AllocQueryContext()` in the library identified by *WRLN* with *FSCH* as input argument.

v) Let *TEMP* be either a <query specification> of the form "SELECT * FROM *FTN*" or a <query specification> *IDQS* of the form "SELECT *DistinctOrAll* exp_1, exp_2, ..., exp_n FROM FTN_1, FTN_2, ..., FTN_m WHERE BVE_1 AND BVE_2 AND ... AND BVE_p", where all of the following are true:

1) *DistinctOrAll* is either "DISTINCT" or "ALL".

2) *n*, *m*, and *p* are implementation-dependent numeric values.

3) For all *i*, 1 (one) ≤ *i* ≤ *n*, exp_i is an implementation-dependent <value expression> that does not generally contain a <query expression> or a <routine invocation> one of whose subject routines possibly reads SQL-data.

4) For all *i*, 1 (one) ≤ *i* ≤ *m*, FTN_i is a foreign table, and at least one of FTN_ishall be equivalent to *FTN*.

5) For all i, 1 (one) ≤ i ≤ m, the table descriptor of FTN_i shall include a foreign server descriptor that is equal to *FSN*.

6) For all i, 1 (one) ≤ i ≤ p, BVE_i is a <boolean value expression> that does not generally contain a <query expression> or a <routine invocation> one of whose subject routines possibly reads SQL-data.

vi) Let *RQH* be the RequestHandle allocated for *TEMP*.

vii) Let *NTR* be the number of <table reference>s in *TEMP*. Let TRH_i, 1 (one) ≤ i ≤ *NTR*, be the TableReferenceHandle allocated for each <table reference> simply contained in *TEMP*.

viii) Let *N* be the number of <value expression>s contained in the <select list> simply contained in *TEMP*. Let CN_i, 1 (one) ≤ i ≤ *N*, be the i-th such <value expression>. Let VEH_i, 1 (one) ≤ i ≤ *N* be the i-th ValueExpressionHandle allocated for CN_i.

ix) Let *M* be the number of <value expression>s contained in the <where clause> simply contained in *TEMP*. Let CN_i, *N*+1 ≤ i ≤ *N*+*M*, be the i-th such <value expression>. Let VEH_i, *N*+1 ≤ i ≤ *N*+*M*, be the i-th ValueExpressionHandle allocated for CN_i.

x) For all i, 1 (one) ≤ i ≤ *N*+*M*, if CN_i identifies an <SQL-invoked routine> whose specific name is *SRN* and there exists a routine mapping descriptor that contains a specific routine name that is equivalent to *SRN* and a foreign server name that is equivalent to *FSN*, then let RH_i be the RoutineMappingHandle allocated for that routine mapping. The resource identified by RH_i is referred to as an *allocated routine mapping description*. RH_i is associated with VEH_i.

xi) For all i, 1 (one) ≤ i ≤ *N*+*M*, if CN_i is a <column reference> of a table reference TRH_j, then VEH_i is associated with TRH_j.

xii) For each CN_i, 1 (one) ≤ i ≤ *N*+*M*, a value expression descriptor is allocated that describes the most specific type and value of CN_i. The value of the TOP_LEVEL_COUNT header field is set to 1 (one). The value of each remaining header field and the value of each field in the contained item descriptor area and subordinate item descriptor field, if any, are implementation-dependent. Let $VEDH_i$ be the ValueExpressionDescriptorHandle associated with the value expression descriptor allocated for CN_i. $VEDH_i$ is associated with VEH_i.

xiii) A table reference descriptor *TRD* is automatically allocated. Each of the fields in *TRD* that have non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", is set to the specified default value. All other fields in *TRD* are initially undefined.

xiv) The General Rules of Subclause 21.4, "Implicit DESCRIBE OUTPUT USING clause", are applied with *TEMP* and *TRD* as *SOURCE* and *DESCRIPTOR*, respectively.

xv) A wrapper parameter descriptor *WPD* is automatically allocated. Each of the fields in *WPD* that have non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", is set to the specified default value. All other fields in *WPD* are initially undefined. Let *WPDH* be the handle associated with *WPD*.

xvi) The General Rules of Subclause 21.3, "Implicit DESCRIBE INPUT USING clause", are applied with *TEMP* and *WPD* as *SOURCE* and *DESCRIPTOR*, respectively.

xvii) Let *TRDH* be the TableReferenceDescriptorHandle allocated for *TRD*.

xviii) Let *RPH* and *EXH* be the ReplyHandle and ExecutionHandle, respectively, returned by the invocation of `AdvanceInitRequest()` in the library identified by *WRLN* with *FSCH* and *RQH*, *QCH* as input arguments.

xix) *TRD* and *WPD* are associated with *EXH*.

xx) Let *NRTR* be the NumberOfTableReferences that would be returned by an invocation of `GetNumReplyTableRefs()` with *RPH* as the ReplyHandle parameter.

xxi) Let TRN_i, 1 (one) ≤ *i* ≤ *NRTR*, be the TableReferenceNumber that would be returned by an invocation of `GetReplyTableRef()` with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.

xxii) Let *NSLE* be the NumberOfSelectListElements that would be returned by an invocation of `GetNumReplySelectElems()` with *RPH* as the ReplyHandle parameter.

xxiii) Let $SELN_i$, 1 (one) ≤ *i* ≤ *NSLE*, be the SelectListElementNumber that would returned by an invocation of `GetReplySelectElem()` with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.

xxiv) Let *NBVE* be the NumberOfBoolVEs that would be returned by an invocation of `GetNumReplyBoolVE()` with *RPH* as the ReplyHandle parameter.

xxv) Let $BVEN_i$, 1 (one) ≤ *i* ≤ *NBVE*, be the BoolVENumber that would returned by an invocation of `GetReplyBoolVE()` with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.

xxvi) Let *RCA* be the ReplyCardinality that would be returned by an invocation of `GetReplyCardinality()` with *RPH* as the ReplyHandle parameter.

xxvii) Let *REFC* be the ReplyExecFirstCost that would be returned by an invocation of `GetReplyFirstCost()` with *RPH* as the ReplyHandle parameter.

xxviii) Let *RTEC* be the ReplyTotalExecCost that would be returned by an invocation of `GetReplyExecCost()` with *RPH* as the ReplyHandle parameter.

xxix) Let *RREC* be the ReplyReExecutionCost that would be returned by an invocation of `GetReplyReExecCost()` with *RPH* as the ReplyHandle parameter.

xxx) It is implementation-dependent whether the `NextReply()` routine with *RPH* as the ReplyHandle parameter is invoked. If the `NextReply()` routine is invoked, then let *RPHN* and *EXHN* be the ReplyHandle and ExecutionHandle, respectively, returned by that invocation; GR 1)a)xx) through GR 1)a)xxx) of this Subclause are applied with *RPHN* and *EXHN* as *RPH* and *EXH*, respectively.

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

xxxii) It is implementation-dependent whether GR 1)a)iv) through GR 1)a)xxxii) of this Subclause are applied once again.

xxxiii) Let *NC* be the value of the COUNT descriptor field that would be returned by invocation of `GetDescriptor()` with *TRDH* as the DescriptorHandle parameter, 0 (zero) as the RecordNumber parameter, and the code for COUNT from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

xxxiv) Let DT_j be the effective data type of the j-th column, for 1 (one) ≤ j ≤ 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 `GetDescriptor()` with *TRDH* 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 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

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

xxxvi) Let TDT_j be the effective data type of the j-th <target specification>, for 1 (one) ≤ j ≤ NC, as represented by the values of the TYPE, LENGTH, OCTET_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 be set by separate invocations of `SetDescriptor()` 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. TYPE either indicates ROW or is one of the code values in Table 15, "Codes used for application data types in SQL/CLI".

xxxvii) For every DT_j and TDT_j, 1 (one) ≤ j ≤ NC:

1) If DT_j is an array data type and TDT_j is not an array locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

2) If DT_j is a multiset data type and TDT_j is not a multiset locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

3) If DT_j is a row data type, then

Case:

A) If TDT_j is not a row data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

B) If TDT_j is a row data type and DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

4) If DT_j and TDT_j are predefined data types, then let *HL* be the programming language in which the invoking SQL-server is written. Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 19.5, "SQL/CLI 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".

Case:

A) If the row that contains the SQL data type corresponding to DT_j in the SQL data type column of the operative data type correspondence table contains "None" in the host data type column, and TDT_j is not a character string type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

B) Otherwise, if DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

5) If DT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

xxxviii) Let *NP* be the value of the COUNT descriptor field that would be returned by invocation of `GetDescriptor()` with *WPDH* as the DescriptorHandle parameter, 0 (zero) as the RecordNumber parameter, and the code for COUNT from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

xxxix) Let PDT_j be the effective data type of the *j*-th column, for 1 (one) ≤ *j* ≤ *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 returned by separate invocations of `GetDescriptor()` with *WPDH* 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 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

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

xli) Let SDT_j be the effective data type of the *j*-th <target specification>, for 1 (one) ≤ *j* ≤ *NP*, as represented by the values of the TYPE, LENGTH, OCTET_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 be set by separate invocations of `SetDescriptor()` 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. TYPE either indicates ROW or is one of the code values in Table 15, "Codes used for application data types in SQL/CLI".

xlii) For every PDT_j and SDT_j, 1 (one) ≤ j ≤ NP:

1) If PDT_j is an array data type and SDT_j is not an array locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

2) If PDT_j is a multiset data type and SDT_j is not a multiset locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

3) If PDT_j is a row data type, then

   Case:

   A) If SDT_j is not a row data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

   B) If SDT_j is a row data type and PDT_j and SDT_j do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

4) If PDT_j and SDT_j are predefined data types, then let *HL* be the programming language in which the invoking SQL-server is written. Let operative data type correspondence table be the data type correspondence table for *HL* as specified in Subclause 19.5, "SQL/CLI 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".

   Case:

   A) If the row that contains the SQL data type corresponding to PDT_j in the SQL data type column of the operative data type correspondence table contains "None" in the host data type column, and SDT_j is not a character string type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

   B) Otherwise, if PDT_j and SDT_j do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

5) If DT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

xliii) For all VEH_i, 1 (one) ≤ i ≤ N+M, let RH_i be the allocated routine mapping description associated with VEH_i, if any. RH_i is deallocated and all its resources are freed.

xliv) VEH_i, 1 (one) ≤ i ≤ $N+M$, is deallocated and all its resources are freed.

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

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

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

xlviii) The result of *TP* is a table that consists of every row returned by the repeated invocation of `Iterate()` in the library identified by *WRLN* with *EXH* as the argument until the return code indicates **No data found**.

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

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

b) Otherwise,

Case:

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

ii) Otherwise,

Case:

1) If *T* is a system-versioned table, then

A) Let *SVS* be the implicit or explicit <query system time period specification>, let *SSTARTCOL* be the system-time period start column of *T* and let *SENDCOL* be the system-time period end column of *T*. Let *DT* be the declared type of *SSTARTCOL*.

B) If *SVS* specifies FOR SYSTEM_TIME AS OF, then let *POTV1* be the value of <point in time 1>. Let *POT1* be the result of CAST (*POTV1* AS *DT*).

C) If *SVS* specifies FOR SYSTEM_TIME BETWEEN or FOR SYSTEM_TIME FROM, then let *POTV1* be the value of <point in time 1> and let *POTV2* be the value of <point in time 2>.

Case:

I) If SYMMETRIC is specified and *POTV1* > *POTV2*, then let *POT1* be the result of CAST (*POTV2* AS *DT*) and let *POT2* be the result of CAST (*POTV1* AS *DT*).

II) Otherwise, let *POT1* be the result of CAST (*POTV1* AS *DT*) and let *POT2* be the result of CAST (*POTV2* AS *DT*).

D) Case:

I) If *SVS* specifies FOR SYSTEM_TIME AS OF, then the result of *TP* is a table that consists of every row *R* of *T* for which the result of (*SSTARTCOL* <= *POT1* AND *SENDCOL* > *POT1*) is *True*.

II) If *SVS* specifies FOR SYSTEM_TIME BETWEEN, then the result of *TP* is a table that consists of every row *R* of *T* for which the result of (*POT1* <= *POT2* AND *SENDCOL* > *POT1* AND *SSTARTCOL* <= *POT2*) is *True*.

III) If *SVS* specifies FOR SYSTEM_TIME FROM, then the result of *TP* is a table that consists of every row *R* of *T* for which the result of (*POT1* < *POT2* AND *SENDCOL* > *POT1* AND *SSTARTCOL* < *POT2*) is *True*.

2) Otherwise, the result of *TP* is a table that consists of every row of *T*.

Conformance Rules

No additional Conformance Rules.

(Blank page)

8 URLs

8.1 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 [RFC3986], except that “localhost” has been omitted from the format of FILE URL. [RFC3986] and [RFC2368] 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>... ]

<hsegment character> ::=
    <uchar>
  | <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>

<commercial at> ::=
  @

<dollar sign> ::=
  $

<exclamation point> ::=
  !
```

Syntax Rules

1) In an SQL-environment, a <url> shall reference the same file, regardless of which component in the SQL-environment is interpreting the <url>.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

(Blank page)

9 Additional common rules

This Clause modifies Clause 9, "Additional common rules", in ISO/IEC 9075-2.

9.1 Retrieval assignment

This Subclause modifies Subclause 9.1, "Retrieval assignment", in ISO/IEC 9075-2.

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) Augment GR 7) If the declared type of *T* is DATALINK, then the value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

9.2 Store assignment

This Subclause modifies Subclause 9.2, "Store assignment", in ISO/IEC 9075-2.

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) Augment GR 3)b) If the declared type of *T* is DATALINK, then the value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

9.3 Result of data type combinations

This Subclause modifies Subclause 9.5, "Result of data type combinations", in ISO/IEC 9075-2.

Function

Specify the result data type of the result of certain combinations of 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 SR 3)f) 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.

9.4 Type precedence list determination

This Subclause modifies Subclause 9.7, "Type precedence list determination", in ISO/IEC 9075-2.

Function

Determine the type precedence list of a given type.

Syntax Rules

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

```
DATALINK
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9.5 Determination of identical values

This Subclause modifies Subclause 9.10, "Determination of identical values", in ISO/IEC 9075-2.

Function

Determine whether two instances of values are identical, that is to say, are occurrences of the same value.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 2)d) If *V1* and *V2* are datalinks, then *V1* is identical to *V2* if and only if the File Reference of *V1* is identical to the File Reference of *V2* and the SQL-Mediated Access Indication of *V1* is identical to the SQL-Mediated Access Indication of *V2*.

Conformance Rules

No additional Conformance Rules.

9.6 Equality operations

This Subclause modifies Subclause 9.11, "Equality operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on operations that involve testing for equality.

Syntax Rules

1) Insert this SR The declared type of an operand of an equality operation shall not be DATALINK-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9.7 Grouping operations

This Subclause modifies Subclause 9.12, "Grouping operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on operations that involve grouping of data.

Syntax Rules

1) Insert this SR The declared type of an operand of a grouping operation shall not be DATALINK-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9.8 Multiset element grouping operations

This Subclause modifies Subclause 9.13, "Multiset element grouping operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on the declared element type of a multiset for operations that involve grouping the elements of a multiset.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert this SR The declared element type of a multiset operand of a multiset element grouping operation shall not be DATALINK-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9.9 Ordering operations

This Subclause modifies Subclause 9.14, "Ordering operations", in ISO/IEC 9075-2.

Function

Specify the prohibitions and restrictions by data type on operations that involve ordering of data.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert this SR The declared type of an operand of an ordering operation shall not be DATALINK-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

(Blank page)

10 Additional common elements

This Clause modifies Clause 10, "Additional common elements", in ISO/IEC 9075-2.

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

Syntax Rules

1) Let *GOPL* be the <generic option list>.

2) No two <generic option>s immediately contained in *GOPL* shall have the same <option name>.

NOTE 35 — The permissible values of <option name> and <option value> are defined by the foreign-data wrapper that deals with the object for which these generic options are being specified.

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.

10.2 <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) Let *m* be the number of <alter generic option>s immediately contained in *AGOPL*. For *j* ranging from 1 (one) to *m*:

   a) Let $AGOP_j$ be the *j*-th <alter generic option> immediately contained in *AGOPL*.

   b) For each $AGOP_j$, if <alter operation> is omitted, then *ADD* is implicit.

   c) Let AOP_j and OPN_j be the <alter operation> and <option name>, respectively, specified or implied by $AGOP_j$.

      Case:

      i) If AOP_j is ADD, then:

         1) <option value> shall be specified and *GOPD* shall not include an <option name> that is equivalent to OPN_j.

         2) *AGOPL* shall not immediately contain any other <alter generic option> that immediately contains an <alter operation> that specifies or implies ADD, and an <option name> that is equivalent to OPN_j.

      ii) If AOP_j is SET, then <option value> shall be specified and *GOPD* shall include an <option name> that is equivalent to OPN_j.

      iii) Otherwise, <option value> shall not be specified and *GOPD* shall include an <option name> that is equivalent to OPN_j.

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*. The ordinal positions of all <option name>s and <option value>s having an ordinal position greater than *i* are reduced by 1 (one).

Conformance Rules

None.

11 Schema definition and manipulation

This Clause modifies Clause 11, "Schema definition and manipulation", in ISO/IEC 9075-2.

11.1 <schema definition>

This Subclause modifies Subclause 11.1, "<schema definition>", in ISO/IEC 9075-2.

Function

Define a schema.

Format

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

11.2 <drop schema statement>

This Subclause modifies Subclause 11.2, "<drop schema statement>", in ISO/IEC 9075-2.

Function

Destroy a schema.

Format

No additional Format items.

Syntax Rules

1) Replace SR 4) If RESTRICT is specified, then *S* shall not contain any persistent base tables, global temporary tables, created local temporary tables, foreign tables, views, domains, assertions, character sets, collations, transliterations, triggers, user-defined types, SQL-invoked routines, roles, or sequence generators, and the <schema name> of *S* shall not be contained in the SQL routine body of any routine descriptor.

NOTE 36 — If CASCADE is specified, then such objects will be dropped by the effective execution of the SQL schema manipulation statements specified in the General Rules of this Subclause.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) Let *T* be the <table name> included in the descriptor of any base table, foreign table, or temporary table included in *S*.

Case:

a) If *T* is a base table or temporary table, then the following <drop table statement> is effectively executed:

```
DROP TABLE T CASCADE
```

b) Otherwise, the following <drop foreign table statement> is effectively executed:

```
DROP FOREIGN TABLE T CASCADE
```

Conformance Rules

No additional Conformance Rules.

11.3 <table definition>

This Subclause modifies Subclause 11.3, "<table definition>", in ISO/IEC 9075-2.

Function

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

Format

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

Syntax Rules

1) Replace SR 11)g)iii) A <column option list> shall immediately contain either a <scope clause> or a <default clause>, or at least one <column constraint definition>, or a <datalink control definition>.

2) Insert after SR 11)g)vi) If *CO* specifies <datalink control definition> *DCS*, then let *COLN* 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 *COLN*.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2) For each <column options> *CO*, if *CO* contains a <datalink control definition> *DCD*, then let *CD* be the column descriptor identified by the <column name> specified in *CO*. The link control options specified in *DCD* are included in the datalink data type descriptor that is included in *CD*.

Conformance Rules

No additional Conformance Rules.

11.4 <unique constraint definition>

This Subclause modifies Subclause 11.7, "<unique constraint definition>", in ISO/IEC 9075-2.

Function

Specify a uniqueness constraint for a table.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert after SR 1) The declared type of no column identified by any <column name> in the <unique column list> shall be DATALINK-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.5 <check constraint definition>

This Subclause modifies Subclause 11.9, "<check constraint definition>", in ISO/IEC 9075-2.

Function

Specify a condition for the SQL-data.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert this SR The <search condition> shall not generally contain a <table reference> that references a foreign table.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.6 <alter column data type clause>

This Subclause modifies Subclause 11.19, "<alter column data type clause>", in ISO/IEC 9075-2.

Function

Change the declared type of a column.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert this GR *D* shall not specify DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.7 <drop column definition>

This Subclause modifies Subclause 11.23, "<drop column definition>", in ISO/IEC 9075-2.

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 GR 2) For each row *R* of *T*, if the value of *C* in *R* is not null, then for every site *DLC* whose value is a constituent of the value of *C* and whose declared type is either DATALINK or some distinct type whose source type is DATALINK, let *EF* be the external file referenced by the value of *DLC*. If *EF* is linked, then *EF* is unlinked.

NOTE 37 — The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the data type descriptor of *DLC*, as specified in Subclause 4.8, "Datalinks".

NOTE 38 — "constituent" is defined in Subclause 4.9, "Columns, fields, and attributes".

Conformance Rules

No additional Conformance Rules.

11.8 <domain definition>

This Subclause modifies Subclause 11.34, "<domain definition>", in ISO/IEC 9075-2.

Function

Define a domain.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert before SR 1) <data type> shall not contain a <datalink control definition>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.9 <assertion definition>

This Subclause modifies Subclause 11.47, "<assertion definition>", in ISO/IEC 9075-2.

Function

Specify an integrity constraint.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert this SR The <search condition> shall not generally contain a <table reference> that references a foreign table.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.10 <user-defined type definition>

This Subclause modifies Subclause 11.51, "<user-defined type definition>", in ISO/IEC 9075-2.

Function

Define a user-defined type.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 2)c)i) If *SDT* is neither a large object type nor a datalink type, then the following SQL-statement is executed without further Access Rule checking:

```
CREATE ORDERING FOR UDTN
ORDER FULL BY
  MAP WITH FUNCTION FNSDT(UDTN)
FOR UDTN
```

Conformance Rules

No additional Conformance Rules.

11.11 <SQL-invoked routine>

This Subclause modifies Subclause 11.60, "<SQL-invoked routine>", in ISO/IEC 9075-2.

Function

Define an SQL-invoked routine.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert before SR 1) Neither <returns type> nor <parameter type> shall contain a <datalink control definition>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.12 <drop routine statement>

This Subclause modifies Subclause 11.62, "<drop routine statement>", in ISO/IEC 9075-2.

Function

Destroy an SQL-invoked routine.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert after SR 4)d)ii) A routine mapping descriptor.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2) Let *RM* be any routine mapping descriptor that includes a specific routine name that is equivalent to *SN*. Let *RMN* be the routine mapping name included in *RM*. The following <drop routine mapping statement> is effectively executed without further Access Rule checking:

```
DROP ROUTINE MAPPING RMN
```

Conformance Rules

No additional Conformance Rules.

11.13 <user-defined cast definition>

This Subclause modifies Subclause 11.63, "<user-defined cast definition>", in ISO/IEC 9075-2.

Function

Define a user-defined cast.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert before SR 1) Neither <source data type> nor <target data type> shall contain a <datalink control definition>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.14 <user-defined ordering definition>

This Subclause modifies Subclause 11.65, "<user-defined ordering definition>", in ISO/IEC 9075-2.

Function

Define a user-defined ordering for a user-defined type.

Format

```
No additional Format items.
```

Syntax Rules

1) Insert after SR 6)a)iii) The declared type of each attribute of *UDT* shall not be DATALINK-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.15 <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) ≤ *i* ≤ *n*:

   a) For all *j*, 1 (one) ≤ *j* ≤ *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) The applicable privileges shall include the USAGE privilege on the foreign-server identified by <foreign server name>.

3) Additional privileges, if any, necessary to execute <foreign table definition> are implementation-defined.

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) ≤ i ≤ 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.

   h) An indication that the table is not insertable-into.

   i) An indication that the table is not updatable.

NOTE 39 — This part of ISO/IEC 9075 currently restricts foreign tables such that they are neither insertable-into nor updatable. Future versions of this part of ISO/IEC 9075 may relax these restrictions.

2) Let *T* be the table described by *FTD*. Let *m* be the number of column descriptors CD_i, 1 (one) ≤ *i* ≤ *m*, included in *FTD*. The row type of *T* consists of *m* fields F_i such that, for all *i*, 1 (one) ≤ *i* ≤ *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 is created that define the privilege SELECT on *T* and SELECT for every column of *T*. These privileges are grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM". The grantee is *A*.

Conformance Rules

1) Without Feature M004, "Foreign data support", conforming SQL language shall not contain a <foreign table definition>.

11.16 <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 10.2, "<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 10.2, "<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) ≤ *i* ≤ *m*, included in *FTD*. The row type of *T* consists of *m* fields F_i such that, for all *i*, 1 (one) ≤ *i* ≤ *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 M004, “Foreign data support”, conforming SQL language shall not contain an <alter foreign table statement>.

11.17 <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

None.

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) Let *T* be the table described by *FTD*. 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.

11.18 <alter basic column definition>

Function

Change the definition of a column of 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 basic 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.

11.19 <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 40 — A <drop basic 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 41 — 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 42 — *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.

11.20 <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 43 — 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 44 — This deletion does not create a new state change in the most recent statement 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 M004, “Foreign data support”, conforming SQL language shall not contain a <drop foreign table statement>.

12 Catalog manipulation

12.1 <foreign server definition>

Function

Define a foreign server.

Format

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

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

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

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) 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>.

2) Additional privileges, if any, necessary to execute <foreign server definition> are implementation-defined.

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 current authorization identifier.

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 current authorization identifier>. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable.

Conformance Rules

1) Without Feature M004, "Foreign data support", conforming SQL language shall not contain a <foreign server definition>.

12.2 <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 10.2, "<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 descriptor identified by *FSN*.

Access Rules

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

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 10.2, "<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 M004, "Foreign data support", conforming SQL language shall not contain an <alter foreign server statement>.

12.3 <drop foreign server statement>

Function

Destroy a foreign server descriptor.

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 of the following:

   a) A foreign table descriptor.

   b) A routine mapping descriptor.

   c) A user mapping descriptor.

3) Let *A* be the authorization identifier that owns the foreign server descriptor 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 *FSN*. 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 FSN
   ```

2) Let *RM* be any routine mapping descriptor that includes a foreign server name that is equivalent to *FSN*. Let *RMN* be the routine mapping name included in *RM*. The following <drop routine mapping statement> is effectively executed without further Access Rule checking:

   ```
   DROP ROUTINE MAPPING RMN
   ```

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

   ```
   REVOKE ALL PRIVILEGES ON FSN FROM A CASCADE
   ```

4) The descriptor *S* is destroyed.

Conformance Rules

1) Without Feature M004, “Foreign data support”, conforming SQL language shall not contain a <drop foreign server statement>.

12.4 <foreign-data wrapper definition>

Function

Define a foreign-data wrapper

Format

```
<foreign-data wrapper definition> ::=
  CREATE FOREIGN DATA WRAPPER <foreign-data wrapper name>
      [ <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 *WN*. *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.

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 current authorization identifier.

   c) The implicit or explicit <library name>.

   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 current authorization identifier. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable.

Conformance Rules

1) Without Feature M004, “Foreign data support”, conforming SQL language shall not contain a <foreign-data wrapper definition>.

12.5 <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 10.2, "<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 descriptor identified by *WN*.

Access Rules

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

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 10.2, "<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 M004, "Foreign data support", conforming SQL language shall not contain an <alter foreign-data wrapper statement>.

12.6 <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 descriptor 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 M004, "Foreign data support", conforming SQL language shall not contain a <drop foreign-data wrapper statement>.

12.7 <import foreign schema statement>

Function

Acquire information about some or all foreign tables associated with a schema managed by a foreign server.

Format

```
<import foreign schema statement> ::=
  IMPORT FOREIGN SCHEMA <foreign schema name> [ <import qualifications> ]
      FROM SERVER <foreign server name> INTO <local schema name>

<import qualifications> ::=
    LIMIT TO <left paren> <table name list> <right paren>
  | EXCEPT <left paren> <table name list> <right paren>

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

<foreign schema name> ::=
  <schema name>

<local schema name> ::=
  <schema name>
```

Syntax Rules

1) Let *FSN* be <foreign schema name>.

2) For every <table name> *TN* contained in <table name list>:

   a) If *TN* specifies a <schema name> *SN*, then *SN* shall be equivalent to *FSN*.

   b) Otherwise, a <schema name> that is equivalent to *FSN* is implicit.

3) There shall be an SQL-schema identified by <local schema name> *LSN*.

Access Rules

None.

General Rules

1) If the foreign server *FSVR* identified by <foreign server name> *FSVRN* does not maintain information analogous to schemas, or if the foreign-data wrapper by which the SQL-server accesses *FSVR* does not support schema importation, then an exception condition is raised: *FDW-specific condition — no schemas*.

2) If *FSVR* does not maintain information about a schema *FS* whose name is equivalent to *FSN*, then an exception condition is raised: *FDW-specific condition — schema not found*.

3) Case:

a) If <import qualifications> is not specified, then let *ITNL* be a <table name list> that contains the <table name> of every table associated with *FS*.

b) If <import qualifications> specifies LIMIT TO, then let *ITNL* be the explicit <table name list>.

c) If <import qualifications> specifies EXCEPT, then let *ITNL* be a <table name list> that contains the <table name> of every table associated with *FS* except the tables whose names are specified in the explicit <table name list>.

4) For every <table name> *FTN* contained in *ITNL*, if *FS* does not include a descriptor of a table whose <table name> is equivalent to *FTN*, then an exception condition is raised: *FDW-specific condition — table not found*.

5) For every <table name> *FTN* contained in *ITNL*:

a) Let *n* be the number of columns whose descriptors are included in the table identified by *FTN*.

b) Let BCD_i, 1 (one) **≤** *i* **≤** *n*, be a <basic column definition> that contains a <column name> equivalent to the name of the *i*-th column *COL* of the table identified by *FTN*, a <data type> corresponding to the data type of *COL*, and implementation-defined <column generic options>.

c) Let *FTD* be a <foreign table definition> that contains *FTN*, every BCD_i, 1 (one) **≤** *i* **≤** *n*, in sequence, separated by <comma>s, *FSVRN*, and implementation-defined <table generic options>.

d) *FTD* is effectively executed.

Conformance Rules

1) Without Feature M004, “Foreign data support”, conforming SQL language shall not contain an <import foreign schema statement>.

2) Without Feature M005, “Foreign schema support”, conforming SQL language shall not specify <import foreign schema statement>.

12.8 <routine mapping definition>

Function

Define a routine mapping.

Format

```
<routine mapping definition> ::=
  CREATE ROUTINE MAPPING <routine mapping name> FOR <specific routine designator>
      SERVER <foreign server name> [ <generic options> ]
```

Syntax Rules

1) Let *FSN* be the <foreign server name>. Let *RMN* be the <routine mapping name>.

2) 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*.

3) The SQL-environment shall not include a routine mapping descriptor whose routine mapping name is *RMN*.

4) Let *R* be the SQL-invoked routine identified by the <specific routine designator>. *R* shall identify an SQL-invoked regular function.

5) Let *SRN* be the <specific name> of *R*.

6) The SQL-environment shall not include a routine mapping descriptor whose specific routine name is *SRN* and whose foreign server name is *FSN*.

Access Rules

1) The applicable privileges shall include the USAGE privilege on the foreign server identified by *FSN*.

2) Additional privileges, if any, necessary to execute <routine mapping definition> are implementation-defined.

General Rules

1) A routine mapping descriptor *RMD* is created. *RMD* includes:

   a) The routine mapping name *RMN*.

   b) The specific routine name *SRN*.

   c) The foreign server name *FSN*.

   d) 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 M004, “Foreign data support”, conforming SQL language shall not contain a <routine mapping definition>.

12.9 <alter routine mapping statement>

Function

Change the definition of a routine mapping.

Format

```
<alter routine mapping statement> ::=
  ALTER ROUTINE MAPPING <routine mapping name> <alter generic options>
```

Syntax Rules

1) Let *RMN* be the <routine mapping name> and let *AGO* be the <alter generic options>.

2) The SQL-environment shall include a routine mapping descriptor *RMD* whose routine mapping name is *RMN*.

3) The Syntax Rules of Subclause 10.2, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *RMD* as the applicable generic options descriptor.

Access Rules

1) The privileges necessary to execute <alter routine mapping statement> are implementation-defined.

General Rules

1) The General Rules of Subclause 10.2, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *RMD* as the applicable generic options descriptor.

Conformance Rules

1) Without Feature M004, "Foreign data support", conforming SQL language shall not contain an <alter routine mapping statement>.

12.10 <drop routine mapping statement>

Function

Destroy a routine mapping.

Format

```
<drop routine mapping statement> ::=
  DROP ROUTINE MAPPING <routine mapping name>
```

Syntax Rules

1) Let *RMN* be the <routine mapping name>.

2) The SQL-environment shall include a routine mapping descriptor *RMD* whose routine mapping name is *RMN*.

Access Rules

1) The privileges necessary to execute <drop routine mapping statement> are implementation-defined.

General Rules

1) *RMD* is destroyed.

Conformance Rules

1) Without Feature M004, “Foreign data support”, conforming SQL language shall not contain a <drop routine mapping statement>.

(Blank page)

13 Access control

This Clause modifies Clause 12, "Access control", in ISO/IEC 9075-2.

13.1 <privileges>

This Subclause modifies Subclause 12.3, "<privileges>", in ISO/IEC 9075-2.

Function

Specify privileges.

Format

```
<object name> ::=
    !! All alternatives from ISO/IEC 9075-2
  | FOREIGN DATA WRAPPER <foreign-data wrapper name>
  | FOREIGN SERVER <foreign server name>
```

Syntax Rules

1) Augment SR 2) 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.

13.2 <revoke statement>

This Subclause modifies Subclause 12.7, "<revoke statement>", in ISO/IEC 9075-2.

Function

Destroy privileges and role authorizations.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 16) Let *T* be any foreign 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 foreign table described by *T*.

2) Insert after GR 30) 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 GR 31) Add abandoned foreign server descriptor and abandoned foreign table descriptor to the list of objects whose existence would cause an exception condition to be raised: *dependent privilege descriptors still exist*.

4) 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
```

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

```
DROP FOREIGN TABLE S1.FTN CASCADE
```

Conformance Rules

No additional Conformance Rules.

13.3 <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
  | PUBLIC
```

Syntax Rules

1) Let *FSN* be the <foreign server name>. If <authorization identifier> is specified, then let *U* be the <authorization identifier>; if PUBLIC is specified, then let *U* be PUBLIC; otherwise, let *U* be the current authorization identifier.

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

3) 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*.

Access Rules

1) The applicable privileges shall include the USAGE privilege on the foreign server identified by *FSN*.

2) Additional privileges, if any, necessary to execute <user mapping definition> are implementation-defined.

General Rules

1) A user mapping descriptor *UMD* is created. *UMD* includes:

   a) Case:

      i) If <specific or generic authorization identifier> specifies PUBLIC, then PUBLIC.

      ii) Otherwise, 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 M004, “Foreign data support”, conforming SQL language shall not contain a <user mapping definition>.

13.4 <alter user mapping statement>

Function

Change the definition of a 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> and let *AGO* be the <alter generic options>. If <authorization identifier> is specified, then let *U* be the <authorization identifier>; if PUBLIC is specified, then let *U* be PUBLIC; otherwise, let *U* be the current authorization identifier.

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

3) The Syntax Rules of Subclause 10.2, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *UMD* as the applicable generic options descriptor.

Access Rules

1) The privileges necessary to execute <alter user mapping statement> are implementation-defined.

General Rules

1) The General Rules of Subclause 10.2, "<alter generic options>", are applied to *AGO* with the generic options descriptor included in *UMD* as the applicable generic options descriptor.

Conformance Rules

1) Without Feature M004, "Foreign data support", conforming SQL language shall not contain an <alter user mapping statement>.

13.5 <drop user mapping statement>

Function

Destroy a 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>. If <authorization identifier> is specified, then let *U* be the <authorization identifier>; if PUBLIC is specified, then let *U* be PUBLIC; otherwise, let *U* be the current authorization identifier.

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

Access Rules

1) The privileges necessary to execute <drop user mapping statement> are implementation-defined.

General Rules

1) *UMD* is destroyed.

Conformance Rules

1) Without Feature M004, "Foreign data support", conforming SQL language shall not contain a <drop user mapping statement>.

14 SQL-client modules

This Clause modifies Clause 13, "SQL-client modules", in ISO/IEC 9075-2.

14.1 <SQL-client module definition>

This Subclause modifies Subclause 13.1, "<SQL-client module definition>", in ISO/IEC 9075-2.

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 GR 4)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:

   a) Let *CH* be the FSConnectionHandle.

   b) The `FreeFSConnection()` routine is invoked with *CH* as the argument.

2) Insert after GR 4)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:

   a) Let *EH* be the WrapperEnvHandle.

   b) The `FreeWrapperEnv()` routine is invoked with *EH* as the argument.

Conformance Rules

No additional Conformance Rules.

14.2 <externally-invoked procedure>

This Subclause modifies Subclause 13.3, "<externally-invoked procedure>", in ISO/IEC 9075-2.

Function

Define an externally-invoked procedure.

Format

No additional Format items.

Syntax Rules

1) Insert before SR 1) <host parameter data type> shall not contain a <datalink control definition>.

2) Insert into SR 10)e)

```
DATA_EXCEPTION_DATALINK_VALUE_EXCEEDS_MAXIMUM_LENGTH:
  constant SQLSTATE_TYPE := "2201D";
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";
DATALINK_EXCEPTION_INVALID_WRITE_TOKEN:
  constant SQLSTATE_TYPE := "HW004";
DATALINK_EXCEPTION_INVALID_DATALINK_CONSTRUCTION:
  constant SQLSTATE_TYPE := "HW005";
DATALINK_EXCEPTION_INVALID_WRITE_PERMISSION_FOR_UPDATE:
  constant SQLSTATE_TYPE := "HW006";
DATALINK_EXCEPTION_REFERENCED_FILE_DOES_NOT_EXIST:
  constant SQLSTATE_TYPE := "HW003";
DATALINK_EXCEPTION_REFERENCED_FILE_NOT_VALID:
  constant SQLSTATE_TYPE := "HW007";
FDW_SPECIFIC_CONDITION_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "HV000";
FDW_SPECIFIC_CONDITION_COLUMN_NAME_NOT_FOUND:
  constant SQLSTATE_TYPE := "HV005";
FDW_SPECIFIC_CONDITION_DYNAMIC_PARAMETER_VALUE_NEEDED:
  constant SQLSTATE_TYPE := "HV002";
FDW_SPECIFIC_CONDITION_FUNCTION_SEQUENCE_ERROR:
  constant SQLSTATE_TYPE := "HV010";
FDW_SPECIFIC_CONDITION_INCONSISTENT_DESCRIPTOR_INFORMATION:
  constant SQLSTATE_TYPE := "HV021";
FDW_SPECIFIC_CONDITION_INVALID_ATTRIBUTE_VALUE:
  constant SQLSTATE_TYPE := "HV024";
FDW_SPECIFIC_CONDITION_INVALID_COLUMN_NAME:
```

```
    constant SQLSTATE_TYPE := "HV007";
FDW_SPECIFIC_CONDITION_INVALID_COLUMN_NUMBER:
    constant SQLSTATE_TYPE := "HV008";
FDW_SPECIFIC_CONDITION_INVALID_DATA_TYPE:
    constant SQLSTATE_TYPE := "HV004";
FDW_SPECIFIC_CONDITION_INVALID_DATA_TYPE_DESCRIPTORS:
    constant SQLSTATE_TYPE := "HV006";
FDW_SPECIFIC_CONDITION_INVALID_DESCRIPTOR_FIELD_IDENTIFIER:
    constant SQLSTATE_TYPE := "HV091";
FDW_SPECIFIC_CONDITION_INVALID_HANDLE:
    constant SQLSTATE_TYPE := "HV00B";
FDW_SPECIFIC_CONDITION_INVALID_OPTION_INDEX:
    constant SQLSTATE_TYPE := "HV00C";
FDW_SPECIFIC_CONDITION_INVALID_OPTION_NAME:
    constant SQLSTATE_TYPE := "HV00D";
FDW_SPECIFIC_CONDITION_INVALID_STRING_FORMAT:
    constant SQLSTATE_TYPE := "HV00A";
FDW_SPECIFIC_CONDITION_INVALID_STRING_LENGTH_OR_BUFFER_LENGTH:
    constant SQLSTATE_TYPE := "HV090";
FDW_SPECIFIC_CONDITION_INVALID_USE_OF_NULL_POINTER:
    constant SQLSTATE_TYPE := "HV009";
FDW_SPECIFIC_CONDITION_LIMIT_ON_NUMBER_OF_HANDLES_EXCEEDED:
    constant SQLSTATE_TYPE := "HV014";
FDW_SPECIFIC_CONDITION_MEMORY_ALLOCATION_ERROR:
    constant SQLSTATE_TYPE := "HV001";
FDW_SPECIFIC_CONDITION_NO_SCHEMAS:
    constant SQLSTATE_TYPE := "HV00P";
FDW_SPECIFIC_CONDITION_OPTION_NAME_NOT_FOUND:
    constant SQLSTATE_TYPE := "HV00J";
FDW_SPECIFIC_CONDITION_REPLY_HANDLE:
    constant SQLSTATE_TYPE := "HV00K";
FDW_SPECIFIC_CONDITION_SCHEMA_NOT_FOUND:
    constant SQLSTATE_TYPE := "HV00Q";
FDW_SPECIFIC_CONDITION_TABLE_NOT_FOUND:
    constant SQLSTATE_TYPE := "HV00R";
FDW_SPECIFIC_CONDITION_UNABLE_TO_CREATE_EXECUTION:
    constant SQLSTATE_TYPE := "HV00L";
FDW_SPECIFIC_CONDITION_UNABLE_TO_CREATE_REPLY:
    constant SQLSTATE_TYPE := "HV00M";
FDW_SPECIFIC_CONDITION_UNABLE_TO_ESTABLISH_CONNECTION:
    constant SQLSTATE_TYPE := "HV00N";
INVALID_FOREIGN_SERVER_SPECIFICATION_NO_SUBCLASS:
    constant SQLSTATE_TYPE := "0X000";
PASSTHROUGH_SPECIFIC_CONDITION_NO_SUBCLASS:
    constant SQLSTATE_TYPE := "0Y000";
PASSTHROUGH_SPECIFIC_CONDITION_INVALID_CURSOR_OPTION:
    constant SQLSTATE_TYPE := "0Y001";
PASSTHROUGH_SPECIFIC_CONDITION_INVALID_CURSOR_ALLOCATION:
    constant SQLSTATE_TYPE := "0Y002";
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

14.3 <SQL procedure statement>

This Subclause modifies Subclause 13.4, "<SQL procedure statement>", in ISO/IEC 9075-2.

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
  | <foreign table definition>
  | <foreign server definition>
  | <foreign-data wrapper definition>
  | <user mapping definition>
  | <routine mapping definition>

<SQL schema manipulation statement> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <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>
  | <alter routine mapping statement>
  | <drop routine mapping statement>

<SQL session statement> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <set passthrough 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.

14.4 Data type correspondences

This Subclause modifies Subclause 13.5, "Data type correspondences", in ISO/IEC 9075-2.

Function

Specify the data type correspondences for SQL data types and host language types.

Tables

Table 5, "Data type correspondences for Ada", modifies Table 19, "Data type correspondences for Ada", in [ISO9075-2].

Table 5 — Data type correspondences for Ada

SQL Data Type	Ada Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	SQL_STANDARD.CHAR, with P'LENGTH of *LD*[1]
[1] The length *LD* of the Ada character 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 4.8, "Datalinks".)	

Table 6, "Data type correspondences for C", modifies Table 20, "Data type correspondences for C", in [ISO9075-2].

Table 6 — Data type correspondences for C

SQL Data Type	C Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	char, with length *LD*[5]
[5] The length *LD* of the C character 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 4.8, "Datalinks".)	

Table 7, "Data type correspondences for COBOL", modifies Table 21, "Data type correspondences for COBOL", in [ISO9075-2].

Table 7 — Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	alphanumeric, with length *LD*[4]
[4] The length *LD* of the COBOL character 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 4.8, "Datalinks".)	

Table 8, "Data type correspondences for Fortran", modifies Table 22, "Data type correspondences for Fortran", in [ISO9075-2].

Table 8 — Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	CHARACTER with length *LD*[4]
[4] The length *LD* of the Fortran character 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 4.8, "Datalinks".)	

Table 9, "Data type correspondences for M", modifies Table 23, "Data type correspondences for M", in [ISO9075-2].

Table 9 — Data type correspondences for M

SQL Data Type	MUMPS Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	character

Table 10, "Data type correspondences for Pascal", modifies Table 24, "Data type correspondences for Pascal", in [ISO9075-2].

Table 10 — Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	PACKED ARRAY[1..*LD*[2]] OF CHAR
[2] The length *LD* of the Pascal character 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 4.8, "Datalinks".)	

Table 11, "Data type correspondences for PL/I", modifies Table 25, "Data type correspondences for PL/I", in [ISO9075-2].

Table 11 — Data type correspondences for PL/I

SQL Data Type	PL/I Data Type
All alternatives from ISO/IEC 9075-2	
DATALINK	CHARACTER VARYING(*LD*[2])
[2] The length *LD* of the PL/I character 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 4.8, "Datalinks".)	

Conformance Rules

No additional Conformance Rules.

15 Additional data manipulation rules

This Clause modifies Clause 15, "Additional data manipulation rules", in ISO/IEC 9075-2.

15.1 Effect of deleting rows from base tables

This Subclause modifies Subclause 15.7, "Effect of deleting rows from base tables", in ISO/IEC 9075-2.

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 GR 7) For each row *R* that is marked for deletion from *T*, for each site *DLC* whose value is a constituent of the value of *R* such that the declared type of *DLC* is DATALINK or some distinct type whose source data type is DATALINK, and such that the data type descriptor of the declared type of *DLC* is or includes a datalink data type descriptor with link control option FILE LINK CONTROL, let *DLCV* be the value of *DLC*.

NOTE 45 — "constituent" is defined in Subclause 4.9, "Columns, fields, and attributes".

2) Insert after GR 7) If *DLCV* is not the null value, then let *EF* be the external file referenced by *DLCV*.

Case:

a) If *EF* is not linked and the integrity control option included in the descriptor of *DLC* specifies INTEGRITY ALL, then an exception condition is raised: *datalink exception — external file not linked*.

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

NOTE 46 — The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in Subclause 4.8, "Datalinks".

Conformance Rules

No additional Conformance Rules.

15.2 Effect of inserting tables into base tables

This Subclause modifies Subclause 15.10, "Effect of inserting tables into base tables", in ISO/IEC 9075-2.

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 after GR 5)c) For each row *R* inserted into *T*, for each site *DLC* whose value is a constituent of the value of *R* such that the declared type of *DLC* is DATALINK or some distinct type whose source type is DATALINK, and such that the data type descriptor of the declared type of *DLC* is or includes a datalink data type descriptor with link control options FILE LINK CONTROL and either INTEGRITY ALL or INTEGRITY SELECTIVE, let *DLCV* be the value of *DLC*.

NOTE 47 — "constituent" is defined in Subclause 4.9, "Columns, fields, and attributes".

a) If *DLCV* is not the null value, then:

Case:

i) If *DLCV* does not reference a file, then an exception condition is raised: *datalink exception — referenced file does not exist*.

ii) Otherwise, let *EF* be the external file referenced by *DLCV*.

b) Case:

i) If the Construction Indication of *DLCV* is not the null value, then an exception condition is raised: *datalink exception — invalid datalink construction*.

ii) If *EF* is linked, then an exception condition is raised: *datalink exception — external file already linked*.

iii) If INTEGRITY ALL is specified, then *EF* is linked according to the <datalink file control option> READ PERMISSION and WRITE PERMISSION of *DLC*.

iv) 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*.

c) Case:

i) If the read permission option included in the descriptor of *DLC* is DB, then the SQL-Mediated Read Access Indication of *DLCV* is set to *True*.

ii) Otherwise, the SQL-Mediated Read Access Indication of *DLCV* is set to *False*.

d) Case:

i) If the write permission option included in the descriptor of *DLC* is either ADMIN REQUIRING TOKEN FOR UPDATE or ADMIN NOT REQUIRING TOKEN FOR UPDATE, then the SQL-Mediated Write Access Indication of *DLCV* is set to *True*.

ii) Otherwise, the SQL-Mediated Write Access Indication of *DLCV* is set to *False*.

e) The Write Token of *DLCV* is set to the null value.

f) The Construction Indication of *DLCV* is set to the null value.

Conformance Rules

No additional Conformance Rules.

15.3 Effect of replacing rows in base tables

This Subclause modifies Subclause 15.13, "Effect of replacing rows in base tables", in ISO/IEC 9075-2.

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 GR 9) For each replaced row *R*, for each site *DLC* whose value is a constituent of the value of *R* such that the declared type of *DLC* is DATALINK or some distinct type source type is DATALINK, and such that the data type descriptor of the declared type of *DLC* is or includes a datalink data type descriptor with link control options FILE LINK CONTROL and either INTEGRITY ALL or INTEGRITY SELECTIVE, let *DLCV1* be the value of *DLC* and let *DLCV2* be the value of the site in the new transition variable that corresponds to *DLC*.

NOTE 48 — "constituent" is defined in Subclause 4.9, "Columns, fields, and attributes".

a) If *DLCV1* is not the null value, then let *EF1* be the external file referenced by *DLCV1*.

Case:

i) If *EF1* is not linked and the integrity control option included in the descriptor of *DLC* specifies INTEGRITY ALL, then an exception condition is raised: *datalink exception — external file not linked*.

ii) If *EF1* is linked, *EF1* is unlinked.

NOTE 49 — The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in Subclause 4.8, "Datalinks".

b) If *DLCV2* is not the null value, then

i) Case:

1) If *DLCV2* does not reference a file, then an exception condition is raised: *datalink exception — referenced file does not exist*.

2) Otherwise, let *EF2* be the external file referenced by *DLCV2*.

ii) Case:

1) If *EF2* is linked, then an exception condition is raised: *datalink exception — external file already linked*.

2) If the Construction Indication of *DLCV2* is either NEWCOPY or PREVIOUSCOPY, then:

A) If the write permission option included in the descriptor of *DLC* is ADMIN REQUIRING TOKEN FOR UPDATE, then

Case:

I) If the Write Token of *DLCV2* is the null value, then an exception condition is raised: *datalink exception — invalid write token*.

II) If the Write Token of *DLCV2* is not valid according to implementation-defined rules, then an exception condition is raised: *datalink exception — invalid write token*.

B) If the write permission option included in the descriptor of *DLC* is BLOCKED, then an exception condition is raised: *datalink exception — invalid write permission for update*.

C) If the File Reference of *DLCV1* and the File Reference of *DLCV2* are not identical, then an exception condition is raised: *datalink exception — referenced file not valid*.

D) *EF2* is linked according to the read permission option and write permission option included in the descriptor of *DLC*.

3) If INTEGRITY ALL is specified, then *EF2* is linked according to the <datalink file control option> of READ PERMISSION and WRITE PERMISSION of *DLC*.

4) If INTEGRITY SELECTIVE is specified, then *EF2* may be linked in an implementation-defined manner according to the <datalink file control option> of READ PERMISSION and WRITE PERMISSION of *DLC*.

iii) Case:

1) If the read permission option included in the descriptor of *DLC* is DB, then the SQL-Mediated Read Access Indication of *DLCV2* is set to *True*.

2) Otherwise, the SQL-Mediated Read Access Indication of *DLCV2* is set to *False*.

iv) Case:

1) If the write permission option included in the descriptor of *DLC* is either ADMIN REQUIRING TOKEN FOR UPDATE or ADMIN NOT REQUIRING TOKEN FOR UPDATE, then the SQL-Mediated Write Access Indication of *DLCV2* is set to *True*.

2) Otherwise, the SQL-Mediated Write Access Indication of *DLCV2* is set to *False*.

v) The Write Token of *DLCV2* is set to the null value.

vi) The Construction Indication of *DLCV2* is set to the null value.

Conformance Rules

No additional Conformance Rules.

16 Session management

This Clause modifies Clause 19, "Session management", in ISO/IEC 9075-2.

16.1 <set passthrough statement>

Function

Set the pass-through flag to *True* or *False* for the current SQL-session context.

Format

```
<set passthrough statement> ::=
  SET PASSTHROUGH <passthrough specification>

<passthrough specification> ::=
    <value specification>
  | OFF
```

Syntax Rules

1) The declared type of the <value specification> shall be a character string type.

Access Rules

None.

General Rules

1) If there is a pass-through foreign server name included in the current SQL-session context, then let *FSN* be that pass-through foreign server name, 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*, and let *WRLN* be the name of the library identified in the foreign-data wrapper descriptor of *WR*.

2) For each execution handle EXH_i that is part of an {<SQL statement name> : ExecutionHandle} pair that is present in the current SQL-session context, the `FreeExecutionHandle()` routine in the library identified by *WRLN* is invoked with EXH_i as the argument.

3) All {<SQL statement name> : ExecutionHandle} pairs present in the current SQL-session context are removed from the current SQL-session context.

4) Case:

a) If <value specification> is specified, then:

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

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

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

   iii) If a foreign server descriptor that includes *V* as the foreign server name exists, then let *FS* be that foreign server. Otherwise, an exception condition is raised: *invalid foreign server specification*.

   iv) If the current privileges do not include USAGE privilege on *FS*, then an exception condition is raised: *invalid foreign server specification*.

   v) The pass-through flag of the current SQL-session context is set to *True*.

   vi) The pass-through foreign server name included in the current SQL-session context is set to the foreign server name of *FS*.

b) Otherwise:

   i) The pass-through flag of the current SQL-session context is set to *False*.

   ii) The pass-through foreign server name included in the current SQL-session context is deleted.

Conformance Rules

1) Without Feature M004, “Foreign data support”, conforming SQL language shall not contain a <set passthrough statement>.

17 Dynamic SQL

This Clause modifies Clause 20, "Dynamic SQL", in ISO/IEC 9075-2.

17.1 Description of SQL descriptor areas

This Subclause modifies Subclause 20.1, "Description of SQL descriptor areas", in ISO/IEC 9075-2.

Function

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

Syntax Rules

1) Insert before SR 6)r) TYPE indicates DATALINK.

2) Insert before SR 7)y) TYPE indicates DATALINK and *T* is specified by DATALINK.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1) Table 12, "Codes used for SQL data types in Dynamic SQL", specifies the codes associated with the SQL data types.

Table 12, "Codes used for SQL data types in Dynamic SQL", modifies Table 28, "Codes used for SQL data types in Dynamic SQL", in [ISO9075-2].

Table 12 — Codes used for SQL data types in Dynamic SQL

Data Type	Code
All alternatives from ISO/IEC 9075-2	*All alternatives from ISO/IEC 9075-2*
DATALINK	70

Conformance Rules

No additional Conformance Rules.

17.2 <prepare statement>

This Subclause modifies Subclause 20.7, "<prepare statement>", in ISO/IEC 9075-2.

Function

Prepare a statement for execution.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True* and if <SQL statement variable> conforms to the Format and Syntax Rules of a <preparable statement> other than <set passthrough statement>, then:

   a) Let *FSN* be the pass-through foreign server name included in the current SQL-session context. 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*.

   b) Case:

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

      ii) Otherwise:

         1) 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*.

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

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

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

c) Case:

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

ii) Otherwise:

1) 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*.

2) If there is a user mapping identified by the current authorization identifier, then let *UH* be the UserHandle allocated for that user mapping; otherwise, let *UH* be the UserHandle allocated for the user mapping identified by PUBLIC. The resource identified by *UH* is referred to as an *allocated user mapping description*.

3) Let *FSCH* be the FSConnectionHandle returned by the invocation of `ConnectServer()` in the library identified by *WRLN* with *WEH*, *SH*, and *UH* as the arguments.

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

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

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

d) Let *STV* be the contents of <SQL statement variable>. Let *STVL* be the length of *STV*. Let *EXH* be the ExecutionHandle returned by the invocation of `TransmitRequest()` in the library identified by *WRLN* with *FSCH*, *STV*, and *STVL* as arguments.

e) If the current SQL-session context includes an {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then let *OEXH* be the ExecutionHandle associated with <SQL statement name>.

i) The `FreeExecutionHandle()` routine in the library identified by *WRLN* is invoked with *OEXH* as the argument.

ii) The {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name> is removed from the current SQL-session context.

f) The {<SQL statement name> : *EXH*} pair is included in the current SQL-session context.

g) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.3 <deallocate prepared statement>

This Subclause modifies Subclause 20.9, "<deallocate prepared statement>", in ISO/IEC 9075-2.

Function

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

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, and the current SQL-session context includes an {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then:

   a) Let *FSN* be the pass-through foreign server name included in the current SQL-session context. 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*. Let *EXH* be the Execution-Handle associated with <SQL statement name>.

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

   c) The {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name> is removed from the current SQL-session context.

   d) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.4 <describe statement>

This Subclause modifies Subclause 20.10, "<describe statement>", in ISO/IEC 9075-2.

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 before GR 1) If the pass-through flag of the current SQL-session context is *True*, and the current SQL-session context includes an {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then:

   a) Let *EXH* be the ExecutionHandle associated with <SQL statement name>. Let *WPD* and *WRD* be the wrapper parameter descriptor and wrapper row descriptor, respectively, associated with the WPDHandle and WRDHandle, respectively, that would be returned by the invocation of `GetWPDHandle()` and `GetWRDHandle()` with *EXH* as the ExecutionHandle parameter.

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

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

   d) *DA* is set as follows:

      i) If the statement being executed is a <describe output statement>, then:

         1) Let *TD* be the value of the COUNT field in *WRD*.

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

         3) All header fields are set to the values of the header fields of *WRD* with the same name.

4) If *TD* is 0 (zero) or *TD* is greater than *N*, then no item descriptor areas are set. Otherwise, the first *TD* item descriptor areas are set with values from the corresponding item descriptor areas and, optionally, subordinate descriptors from *WRD*.

ii) If the statement being executed is a <describe input statement>, then:

1) Let *TD* be the value of the COUNT field in *WPD*.

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

3) All header fields are set to the values of the header fields of *WPD* with the same name.

4) If *TD* is 0 (zero) or *TD* is greater than *N*, then no item descriptor areas are set. Otherwise, the first *TD* item descriptor areas are set with values from the corresponding item descriptor areas and, optionally, subordinate descriptors from *WPD*.

e) No further General Rules of this Subclause are applied.

2) Insert after GR 7)d)xi) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum datalink length.

NOTE 50 — The term “maximum datalink length” is defined in Subclause 4.8, “Datalinks”.

Conformance Rules

No additional Conformance Rules.

17.5 <input using clause>

This Subclause modifies Subclause 20.11, "<input using clause>", in ISO/IEC 9075-2.

Function

Supply input values for an <SQL dynamic statement>.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 1) If the pass-through flag of the current SQL-session context is *True*, then:

   a) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

   b) Case:

      i) If an <input using clause> is used in a <dynamic open statement>, then let *SN* be the <statement name> of the associated <dynamic declare cursor>.

      ii) Otherwise, let *SN* be the <SQL statement name> of the <execute statement>.

   c) Let *EXH* be the ExecutionHandle associated with *SN*. Let *WPD* and *SPD* be the wrapper parameter descriptor and server parameter descriptor, respectively, associated with the WPDHandle and SPDHandle, respectively, that would be returned be the invocation of `GetWPDHandle()` and `GetSPDHandle()` with *EXH* as the ExecutionHandle parameter.

   d) Let *D* be the value of COUNT in *WPD*.

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

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

      i) Let *DA* be the descriptor area identified by <descriptor name>.

      ii) Let *N* be the value of COUNT in *DA*.

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

iv) If the first *N* item descriptor areas in *DA* are not valid as specified in Subclause 17.1, "Description of SQL descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

v) In the first *N* item descriptor areas in *DA*:

1) 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*.

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

g) For 1 (one) ≤ *i* ≤ *D*:

i) Let *TDT* be the effective declared type of the *i*-th input <dynamic parameter specification> defined by the type representation of the corresponding item descriptor area and its subordinate descriptor areas in *WPD*.

ii) Case:

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

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

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

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

Case:

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

II) Otherwise,

Case:

1) 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*.

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

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

iii) Case:

1) If *SDT* is a locator type, then

Case:

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

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

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

Case:

A) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.13, "<cast specification>", in [ISO9075-2], 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_i of the *i*-th input dynamic parameter.

B) Otherwise:

I) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.13, "<cast specification>", in [ISO9075-2], 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 6.13, "<cast specification>", in [ISO9075-2], then an exception condition is raised in accordance with the General Rules of Subclause 6.13, "<cast specification>", in [ISO9075-2].

III) The <cast specification>

```
CAST ( SV AS TDT )
```

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

iv) Case:

1) If <using input descriptor> is specified, then all fields, except DATA and DATA_POINTER, in the *i*-th item descriptor area of *SPD*, that can be set according to Table 33, "Ability to set foreign-data wrapper descriptor fields", are set with values from the corresponding fields of the item descriptor area and, optionally, subordinate descriptors of *DA*.

2) If <using arguments> is specified, then all fields, except DATA and DATA_POINTER, in the *i*-th item descriptor area of *SPD*, that can be set according to Table 33, "Ability to set foreign-data wrapper descriptor fields", are set to implementation-dependent values.

v) Case:

1) If *HL1* and *HL2* are both pointer-supporting languages, then the DATA_POINTER field in the *i*-th item descriptor area of *SPD* is set to the address of the buffer that contains the value TV_i.

2) Otherwise, the DATA field in the *i*-th item descriptor area of *SPD* is set to TV_i.

h) All header fields in *SPD*, that can be set according to Table 33, "Ability to set foreign-data wrapper descriptor fields", are set to the values of the header fields of *WPD* with equivalent names.

i) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.6 <output using clause>

This Subclause modifies Subclause 20.12, "<output using clause>", in ISO/IEC 9075-2.

Function

Supply output variables for an <SQL dynamic statement>.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, then:

   a) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

   b) Case:

      i) If an <output using clause> is used in a <dynamic fetch statement>, then let *SN* be the <statement name> of the associated <dynamic declare cursor>.

      ii) Otherwise, let *SN* be the <SQL statement name> of the <execute statement>.

   c) Let *EXH* be the ExecutionHandle associated with *SN*. Let *WRD* and SRD be the wrapper row descriptor and server row descriptor, respectively, associated with the WRDHandle and SRDHandle, respectively, that would be returned by the invocation of `GetWRDHandle()` and `GetSRDHandle()` with *EXH* as the ExecutionHandle parameter.

   d) Let *D* be the value of COUNT in *WRD*.

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

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

      i) Let *DA* be the descriptor area identified by <descriptor name>.

      ii) Let *N* be the value of COUNT in *DA*.

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

iv) If the first *N* item descriptor areas in *DA* are not valid as specified in Subclause 17.1, "Description of SQL descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

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

g) For 1 (one) ≤ *i* ≤ *D*:

i) Let *SDT* be the effective declared type of *i*-th descriptor area whose LEVEL value is 0 (zero) and its subordinate descriptor areas in *WRD*.

Case:

1) If *HL1* and *HL2* are both pointer-supporting languages, then let *SV* be the value of the buffer addressed by the DATA_POINTER field in the corresponding item descriptor area of *SRD*, with data type *SDT*.

2) Otherwise, let *SV* be the value of the DATA field in the corresponding item descriptor area of *SRD*, with data type *SDT*.

ii) Case:

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

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

B) Let *TDT* be the declared type represented by *IDA*.

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

iii) If the <output using clause> is used in a <dynamic fetch statement>, then let *CR* be the dynamic cursor identified by the <dynamic fetch statement>, and let *LTDT* be the most specific type of the *i*-th <target specification> or <into argument> on the most recently executed <dynamic fetch statement> prior to the current execution, if any, for *CR*. It is implementation-defined whether or not an exception condition is raised: *dynamic SQL error — restricted data type attribute violation* if any of the following are true:

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

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

3) *LTDT* and *TDT* both identify an array type and only one of *LTDT* and *TDT* is an array locator.

4) *LTDT* and *TDT* both identify a multiset type and only one of *LTDT* and *TDT* is a multiset locator.

5) *LTDT* and *TDT* both identify a user-defined type and only one of *LTDT* and *TDT* is a user-defined type locator.

iv) Case:

1) If *TDT* is a locator type, then

Case:

A) If *SV* is not the null value, then a locator *L* that uniquely identifies *SV* is generated and is the value TV_i of the *i*-th <target specification>.

B) Otherwise, the value TV_i of the *i*-th <target specification> is the null value.

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

Case:

A) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.13, "<cast specification>", in [ISO9075-2], 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_i of the *i*-th <target specification>.

B) Otherwise:

I) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.13, "<cast specification>", in [ISO9075-2], 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 6.13, "<cast specification>", in [ISO9075-2], then an exception condition is raised in accordance with the General Rules of Subclause 6.13, "<cast specification>", in [ISO9075-2].

III) The <cast specification>

```
CAST ( SV AS TDT )
```

is effectively performed and the result is the value TV_i of the *i*-th <target specification>.

v) Case:

1) If <into descriptor> is specified, then all fields in *IDA* are set with values from the corresponding fields of the item descriptor area and, optionally, subordinate descriptors of *SRD*.

2) If <into arguments> is specified, then the Rules in Subclause 9.1, "Retrieval assignment", are applied to TV_i and the *i*-th <into argument> as *VALUE* and *TARGET*, respectively.

h) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.7 <execute statement>

This Subclause modifies Subclause 20.13, "<execute statement>", in ISO/IEC 9075-2.

Function

Associate input SQL parameters and output targets with a prepared statement and execute the statement.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, and the current SQL-session context includes an {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then:

   a) Let *FSN* be the pass-through foreign server name included in the current SQL-session context. 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*. Let *EXH* be the Execution-Handle associated with <SQL statement name>.

   b) If a <parameter using clause> is specified, then the General Rules specified in Subclause 17.5, "<input using clause>", for a <parameter using clause> in an <execute statement> are applied.

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

   d) If a <result using clause> is specified, then the General Rules specified in Subclause 17.6, "<output using clause>", for a <result using clause> in an <execute statement> are applied.

   e) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.8 <dynamic declare cursor>

This Subclause modifies Subclause 20.15, "<dynamic declare cursor>", in ISO/IEC 9075-2.

Function

Declare a declared dynamic cursor to be associated with a <statement name>, which may in turn be associated with a <cursor specification>.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, then:

   a) If <cursor sensitivity> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option*.

   b) If <cursor scrollability> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option*.

   c) If <cursor holdability> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option*.

   d) If <cursor returnability> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option*.

   e) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.9 <allocate extended dynamic cursor statement>

This Subclause modifies Subclause 20.17, "<allocate extended dynamic cursor statement>", in ISO/IEC 9075-2.

Function

Define a cursor based on a prepared statement for a <cursor specification>.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, then an exception condition is raised: *pass-through specific condition — invalid cursor allocation*.

Conformance Rules

No additional Conformance Rules.

17.10 <allocate received cursor statement>

This Subclause modifies Subclause 20.18, "<allocate received cursor statement>", in ISO/IEC 9075-2.

Function

Assign a cursor to the ordered set of result sets returned from an SQL-invoked procedure.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, then an exception condition is raised: *pass-through specific condition — invalid cursor allocation*.

Conformance Rules

No additional Conformance Rules.

17.11 <dynamic open statement>

This Subclause modifies Subclause 20.19, "<dynamic open statement>", in ISO/IEC 9075-2.

Function

Associate input dynamic parameters with a <cursor specification> and open the dynamic cursor.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1)a) If the pass-through flag of the current SQL-session context is *True*, and the current SQL-session context includes an {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then:

   a) Let *FSN* be the name of the pass-through foreign server included in the current SQL-session context. 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*. Let *EXH* be the Execution-Handle associated with <SQL statement name>.

   b) If an <input using clause> is specified, then the General Rules specified in Subclause 17.5, "<input using clause>", for <dynamic open statement> are applied.

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

   d) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.12 <dynamic fetch statement>

This Subclause modifies Subclause 20.20, "<dynamic fetch statement>", in ISO/IEC 9075-2.

Function

Fetch a row for a dynamic cursor.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1) If the pass-through flag of the current SQL-session context is *True*, and the current SQL-session context includes a {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then:

   a) Let *FSN* be the pass-through foreign server name included in the current SQL-session context. 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*. Let *EXH* be the Execution-Handle associated with <SQL statement name>.

   b) The `Iterate()` routine in the library identified by *WRLN* is invoked with *EXH* as argument.

   c) The General Rules of Subclause 17.6, "<output using clause>", are applied to the <dynamic fetch statement> and the current row of *CR* as the retrieved row.

   d) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

17.13 <dynamic close statement>

This Subclause modifies Subclause 20.22, "<dynamic close statement>", in ISO/IEC 9075-2.

Function

Close a dynamic cursor.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 3) If the pass-through flag of the current SQL-session context is *True*, and the current SQL-session context includes an {SQL statement name : ExecutionHandle} pair whose SQL statement name is equivalent to <SQL statement name>, then:

   a) Let *FSN* be the pass-through foreign server name included in the current SQL-session context. 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*. Let *EXH* be the Execution-Handle associated with <SQL statement name>.

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

   c) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

18 Embedded SQL

This Clause modifies Clause 21, "Embedded SQL", in ISO/IEC 9075-2.

18.1 <embedded SQL Ada program>

This Subclause modifies Subclause 21.3, "<embedded SQL Ada program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL Ada program>.

Format

```
<Ada derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <Ada DATALINK variable>

<Ada DATALINK variable> ::=
  SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR 5)n) 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 51 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain an <Ada DATALINK variable>.

2) Without Feature M011, “Datalinks via Ada”, conforming SQL language shall not contain an <Ada DATALINK variable>.

18.2 <embedded SQL C program>

This Subclause modifies Subclause 21.4, "<embedded SQL C program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL C program>.

Format

```
<C derived variable> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <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 SR 5)p) 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 52 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, "Datalinks via Embedded SQL", conforming SQL language shall not contain a <C DATALINK variable>.

2) Without Feature M012, "Datalinks via C", conforming SQL language shall not contain a <C DATALINK variable>.

18.3 <embedded SQL COBOL program>

This Subclause modifies Subclause 21.5, "<embedded SQL COBOL program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL COBOL program>.

Format

```
<COBOL derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <COBOL DATALINK variable>

<COBOL DATALINK variable> ::=
  [ USAGE [ IS ] ] SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR 5)m) 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 53 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, "Datalinks via Embedded SQL", conforming SQL language shall not contain a <COBOL DATALINK variable>.

2) Without Feature M013, "Datalinks via COBOL ", conforming SQL language shall not contain a <COBOL DATALINK variable>.

18.4 <embedded SQL Fortran program>

This Subclause modifies Subclause 21.6, "<embedded SQL Fortran program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL Fortran program>.

Format

```
<Fortran derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <Fortran DATALINK variable>

<Fortran DATALINK variable> ::=
  SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR 6)n) 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 54 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, "Datalinks via Embedded SQL", conforming SQL language shall not contain a <Fortran DATALINK variable>.

2) Without Feature M014, "Datalinks via Fortran", conforming SQL language shall not contain a <Fortran DATALINK variable>.

18.5 <embedded SQL MUMPS program>

This Subclause modifies Subclause 21.7, "<embedded SQL MUMPS program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL MUMPS program>.

Format

```
<MUMPS derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <MUMPS DATALINK variable>

<MUMPS DATALINK variable> ::=
  SQL TYPE IS <datalink type>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, "Datalinks via Embedded SQL", conforming SQL language shall not contain a <MUMPS DATALINK variable>.

2) Without Feature M015, "Datalinks via M ", conforming SQL language shall not contain a <MUMPS DATALINK variable>.

18.6 <embedded SQL Pascal program>

This Subclause modifies Subclause 21.8, "<embedded SQL Pascal program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL Pascal program>.

Format

```
<Pascal derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <Pascal DATALINK variable>

<Pascal DATALINK variable> ::=
  SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR 5)n) The syntax

   ```
   SQL TYPE IS <datalink type>
   ```

   shall be replaced by

   ```
   PACKED ARRAY [1..MDL] OF CHAR
   ```

   where *MDL* is the maximum datalink length, in any <Pascal DATALINK variable>.

   NOTE 55 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, "Datalinks via Embedded SQL", conforming SQL language shall not contain a <Pascal DATALINK variable>.

2) Without Feature M016, "Datalinks via Pascal", conforming SQL language shall not contain a <Pascal DATALINK variable>.

18.7 <embedded SQL PL/I program>

This Subclause modifies Subclause 21.9, "<embedded SQL PL/I program>", in ISO/IEC 9075-2.

Function

Specify an <embedded SQL PL/I program>.

Format

```
<PL/I derived type specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <PL/I DATALINK variable>

<PL/I DATALINK variable> ::=
  SQL TYPE IS <datalink type>
```

Syntax Rules

1) Insert after SR 5)n) 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 56 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Without Feature M003, "Datalinks via Embedded SQL", conforming SQL language shall not contain a <PL/I DATALINK variable>.

2) Without Feature M017, "Datalinks via PL/I", conforming SQL language shall not contain a <PL/I DATALINK variable>.

19 Call-Level Interface specifications

This Clause modifies Clause 5, "Call-Level Interface specifications", in ISO/IEC 9075-3.

19.1 <CLI routine>

This Subclause modifies Subclause 5.1, "<CLI routine>", in ISO/IEC 9075-3.

Function

Describe a generic SQL/CLI routine.

Format

```
<CLI routine> ::=
    !! All alternatives from ISO/IEC 9075-3
  | BuildDataLink
  | GetDataLinkAttr
```

Syntax Rules

1) *Table 13, "Abbreviated SQL/CLI generic names", modifies Table 4, "Abbreviated SQL/CLI generic names", in ISO/IEC 9075-3.*

Table 13 — 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

This Subclause modifies Subclause 5.9, "Implicit DESCRIBE USING clause", in ISO/IEC 9075-3.

Function

Specify the rules for an implicit DESCRIBE USING clause.

General Rules

1) Insert after GR 5)c)iv)10) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.

2) Insert after GR 8)d)vi)10) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.

19.3 Description of CLI item descriptor areas

This Subclause modifies Subclause 5.18, "Description of CLI item descriptor areas", in ISO/IEC 9075-3.

Function

Specify the identifiers, data types and codes for fields used in CLI item descriptor areas.

Syntax Rules

1) Insert after SR 5)c)xiv) TYPE indicates DATALINK.

2) Replace SR 7)c)iv) TYPE indicates DEFAULT, CHARACTER, CHARACTER LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY, BINARY VARYING, BINARY LARGE OBJECT, BINARY LARGE OBJECT LOCATOR, SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, USER-DEFINED TYPE LOCATOR, REF, or DATALINK.

3) Insert after SR 12)c)ix) TYPE indicates DATALINK and one of the following is true:

   a) NULL is true.

   b) DEFERRED is true.

4) Replace SR 13)c)iv) TYPE indicates DEFAULT, CHARACTER, CHARACTER LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY, BINARY VARYING, BINARY LARGE

OBJECT, BINARY LARGE OBJECT LOCATOR, SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, USER-DEFINED TYPE LOCATOR, REF, or DATALINK.

General Rules

1) *Table 14, "Codes used for implementation data types in SQL/CLI", modifies Table 7, "Codes used for implementation data types in SQL/CLI", in ISO/IEC 9075-3.*

Table 14 — Codes used for implementation data types in SQL/CLI

Data Type	Code
	All alternatives from ISO/IEC 9075-3
DATALINK	70

2) *Table 15, "Codes used for application data types in SQL/CLI", modifies Table 8, "Codes used for application data types in SQL/CLI", in ISO/IEC 9075-3.*

Table 15 — Codes used for application data types in SQL/CLI

Data Type	Code
	All alternatives from ISO/IEC 9075-3
DATALINK	70

19.4 Other tables associated with CLI

This Subclause modifies Subclause 5.19, "Other tables associated with CLI", in ISO/IEC 9075-3.

Table 16, "Codes used to identify SQL/CLI routines", modifies Table 28, "Codes used to identify SQL/CLI routines", in ISO/IEC 9075-3.

Table 16 — Codes used to identify SQL/CLI routines

Generic Name	Code
	All alternatives from ISO/IEC 9075-3
BuildDataLink	1029
GetDataLinkAttr	1034

Table 17, "Codes and data types for implementation information", modifies Table 29, "Codes and data types for implementation information", in ISO/IEC 9075-3.

Table 17 — 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 18 — Codes used for datalink attributes

Attribute	Code
URL COMPLETE	3
URL PATH	4
URL PATH ONLY	5
URL SCHEME	6
URL SERVER	7
Implementation-defined datalink attribute	< 0

Table 19, "Data types of attributes", modifies Table 20, "Data types of attributes", in ISO/IEC 9075-3.

Table 19 — Data types of attributes

Attribute	Data type	Values
	All alternatives from ISO/IEC 9075-3	
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 scheme
URL SERVER	CHARACTER VARYING(*L*)[1]	Datalink URL server

Attribute	Data type	Values
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 4.8, "Datalinks".)

Conformance Rules

No additional Conformance Rules.

19.5 SQL/CLI data type correspondences

This Subclause modifies Subclause 5.20, "SQL/CLI data type correspondences", in ISO/IEC 9075-3.

Function

Replace 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 [ISO9075-3].

Tables

Table 20, "SQL/CLI data type correspondences for Ada", modifies Table 40, "SQL/CLI data type correspondences for Ada", in ISO/IEC 9075-3.

Table 20 — 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*
DATALINK	SQL_STANDARD.CHAR, with P'Length of *LD*[1]
[1] The length *LD* of the Ada character type corresponding with SQL data type DATALINK is implementation-defined.	

Table 21, "SQL/CLI data type correspondences for C", modifies Table 41, "SQL/CLI data type correspondences for C", in ISO/IEC 9075-3.

Table 21 — 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*
DATALINK	char, with length *LD*[3]
[3] The length *LD* of the C character type corresponding with SQL data type DATALINK is implementation-defined.	

Table 22, "SQL/CLI data type correspondences for COBOL", modifies Table 42, "SQL/CLI data type correspondences for COBOL", in ISO/IEC 9075-3.

Table 22 — 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*

SQL Data Type	COBOL Data Type
DATALINK	alphanumeric, with length *LD*[3]
[3] The length *LD* of the COBOL character type corresponding with SQL data type DATALINK is implementation-defined.	

Table 23, "SQL/CLI data type correspondences for Fortran", modifies Table 43, "SQL/CLI data type correspondences for Fortran", in ISO/IEC 9075-3.

Table 23 — 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*
DATALINK	CHARACTER with length *LD*[2]
[2] The length *LD* of the Fortran character type corresponding with SQL data type DATALINK is implementation-defined.	

Table 24, "SQL/CLI data type correspondences for M", modifies Table 44, "SQL/CLI data type correspondences for M", in ISO/IEC 9075-3.

Table 24 — SQL/CLI data type correspondences for M

SQL Data Type	MUMPS Data Type
All alternatives from ISO/IEC 9075-3	*All alternatives from ISO/IEC 9075-3*
DATALINK	character

Table 25, "SQL/CLI data type correspondences for Pascal", modifies Table 45, "SQL/CLI data type correspondences for Pascal", in ISO/IEC 9075-3.

Table 25 — 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*
DATALINK	PACKED ARRAY[1..*LD*[2]] OF CHAR
[2] The length *LD* of the Pascal character type corresponding with SQL data type DATALINK is implementation-defined.	

Table 26, "SQL/CLI data type correspondences for PL/I", modifies Table 46, "SQL/CLI data type correspondences for PL/I", in ISO/IEC 9075-3.

Table 26 — 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*
DATALINK	CHARACTER VARYING(*LD*), where *LD* is implementation-defined

20 SQL/CLI routines

This Clause modifies Clause 6, "SQL/CLI routines", in ISO/IEC 9075-3.

20.1 BuildDataLink

Function

Build a datalink value.

Definition

```
BuildDataLink (
    StatementHandle       IN  INTEGER,
    DataLocation          IN  CHARACTER(L1),
    DataLocationLength    IN  INTEGER,
    DataLink              OUT CHARACTER(L2),
    BufferLength          IN  INTEGER,
    StringLength          OUT INTEGER )
RETURNS SMALLINT
```

where *L1* has a maximum value equal to the implementation-defined maximum length of a variable-length character string and *L2* 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 57 — *SH* is used only if BuildDataLink issues a completion or exception condition.

2) Let *DL* be the datalink value whose File Reference is DataLocation.

   NOTE 58 — *File Reference* is defined in Subclause 4.8, "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 59 — The term "maximum datalink length" is defined in Subclause 4.8, "Datalinks".

5) Apply the General Rules of Subclause 5.14, "Character string retrieval", in [ISO9075-3] with DataLink, *DL*, BufferLength, and StringLength as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M002, “Datalinks via SQL/CLI”, conforming SQL language shall not contain `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* 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 60 — *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 18, "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 *DL* be the first *DLL* octets of Datalink.

   b) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

6) Let *ML* be the implementation-defined maximum length in characters of a datalink value.

7) If *DL* is not a valid datalink value, then an exception condition is raised: *CLI-specific condition — invalid datalink value*.

8) Let *BL* be the value of BufferLength.

9) 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 19, "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 5.14, "Character string retrieval", in [ISO9075-3] are applied with Value, *AV*, *BL*, and StringLength as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

10) Case:

a) If *A* indicates URL COMPLETE, then *AV* is set to the value of the File Reference of *DL*.

b) If *A* indicates URL PATH, then *AV* is set to the value of the path of *DL*, possibly combined with an access token under the General Rules of Subclause 6.4, "<string value function>".

c) If *A* indicates URL PATH ONLY, then *AV* is set to the value of the path of *DL*.

d) If *A* indicates URL SCHEME, then *AV* is set to the value of the scheme of *DL*.

e) If *A* indicates URL SERVER, then *AV* is set to the value of the host of *DL*.

NOTE 61 — "host", "scheme", and "path" are defined in Subclause 6.6, "<datalink value function>".

11) The General Rules of Subclause 5.14, "Character string retrieval", in [ISO9075-3] are applied with Value, *AV*, *BL*, and StringLength as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M002, "Datalinks via SQL/CLI", conforming SQL language shall not contain `GetDataLinkAttr()`.

20.3 GetInfo

This Subclause modifies Subclause 6.38, "GetInfo", in ISO/IEC 9075-3.

Function

Get information about the implementation.

Definition

```
No additional Definition.
```

General Rules

1) Insert into the dashed list in GR 10)

   — MAXIMUM DATALINK LENGTH

Conformance Rules

1) Without Feature M002, "Datalinks via SQL/CLI", in conforming SQL language, the value of InfoType shall not indicate MAXIMUM DATALINK LENGTH.

(Blank page)

21 SQL/MED common specifications

21.1 Description of foreign-data wrapper item descriptor areas

Function

Specify the identifiers, data types and codes for fields used in foreign-data wrapper item descriptor areas.

Syntax Rules

1) A foreign-data wrapper item descriptor area consists of the fields specified in Table 4, "Fields in foreign-data wrapper descriptor areas".

2) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

3) Given a foreign-data wrapper item descriptor area *IDA* in which the value of LEVEL is some value *N*, the immediately subordinate descriptor areas of *IDA* are those foreign-data wrapper item descriptor areas in which the value of LEVEL is *N*+1 and whose position in the foreign-data wrapper descriptor area follows that of *IDA* and precedes that of any foreign-data wrapper item descriptor area in which the value of LEVEL is less than *N*+1. The subordinate descriptor areas of *IDA* are those foreign-data wrapper item descriptor areas that are immediately subordinate descriptor areas of *IDA* or that are subordinate descriptor areas of a foreign-data wrapper item descriptor area that is immediately subordinate to *IDA*.

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

   Case:

   a) If *DT* is 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 or MULTISET, then the descriptor of the associated element type of *DT*. The subordinate descriptors of *DE* are those descriptors that are immediately subordinate descriptors of *DE* or that are subordinate descriptors of a descriptor that is immediately subordinate to *DE*.

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

   a) SDE_1 is in the first ordinal position.

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

6) Let *HL* be the 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 19.5, "SQL/CLI 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*.

7) A foreign-data wrapper item descriptor area *IDA* in a foreign-data wrapper descriptor area that is a server row descriptor or a server parameter descriptor is *consistent* if and only if all of the following are true:

   a) TYPE indicates ROW or is one of the code values in Table 15, "Codes used for application data types in SQL/CLI".

   b) 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 DEFAULT, CHARACTER, CHARACTER LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY, BINARY VARYING, BINARY LARGE OBJECT, BINARY LARGE OBJECT LOCATOR, SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, USER-DEFINED TYPE LOCATOR, REF, or DATALINK.

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

      v) TYPE indicates DECFLOAT, and PRECISION is a valid precision value for the DECFLOAT data type.

      vi) TYPE indicates ROW and, where *N* is the value of the DEGREE field, there are exactly *N* immediately subordinate descriptor areas of *IDA*, and those item descriptor areas are valid.

      vii) TYPE indicates ARRAY LOCATOR or MULTISET LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that item descriptor area is valid.

      viii) TYPE indicates an implementation-defined data type.

8) Let *IDA* be a foreign-data wrapper item descriptor area in a server parameter descriptor.

9) If the value of INDICATOR is the appropriate 'Code' for SQL NULL DATA in Table 27, "Miscellaneous codes used in CLI", in [ISO9075-3], then NULL is true for *IDA*. Otherwise, NULL is false for *IDA*.

10) *IDA* is *valid* if and only if:

   a) TYPE is one of the code values in Table 15, "Codes used for application data types in SQL/CLI", or TYPE indicates ROW.

   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 server parameter descriptor.

   c) One of the following is true:

      Case:

      i) TYPE indicates CHARACTER, CHARACTER LARGE OBJECT, BINARY, BINARY VARYING, or BINARY LARGE OBJECT, and the value *V* of OCTET_LENGTH is greater than zero, and

         Case:

1) If *HL1* and *HL2* are both pointer-supporting languages, then 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, BINARY, BINARY VARYING, or BINARY LARGE OBJECT data type, as indicated by TYPE.

2) Otherwise, the number of characters wholly contained in the first *V* octets of DATA is a valid length value for a CHARACTER, CHARACTER LARGE OBJECT, BINARY, BINARY VARYING, or BINARY LARGE OBJECT data type, as indicated by TYPE.

ii) TYPE indicates CHARACTER LARGE OBJECT LOCATOR, BINARY LARGE OBJECT LOCATOR, or USER-DEFINED TYPE LOCATOR.

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

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

v) TYPE indicates SMALLINT, INTEGER, BIGINT, REAL, or DOUBLE PRECISION.

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

vii) TYPE indicates DECFLOAT, and PRECISION is a valid precision value for the DECFLOAT data type.

viii) TYPE indicates REF.

ix) TYPE indicates DATALINK.

x) TYPE indicates ROW and, where *N* is the value of the DEGREE field, there are exactly *N* immediately subordinate foreign-data wrapper descriptor areas of *IDA*, and those foreign-data wrapper item descriptor areas are valid.

xi) TYPE indicates ARRAY LOCATOR or MULTISET LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that subordinate descriptor area is valid.

xii) TYPE indicates an implementation-defined data type.

d) Case:

i) If *HL1* and *HL2* are both pointer-supporting languages, then one of the following is true:

1) DATA_POINTER is zero and NULL is true.

2) DATA_POINTER is not zero and the value of the host variable addressed by DATA_POINTER is a valid value of the data type indicated by TYPE.

ii) Otherwise, DATA is a valid value of the data type indicated by TYPE.

11) A foreign-data wrapper item descriptor area *IDA* in a server row descriptor is valid if and only if:

a) TYPE is one of the code values in Table 15, “Codes used for application data types in SQL/CLI”, or TYPE indicates ROW.

b) If LEVEL is 0 (zero) for *IDA*, then let *TLC* be the value of TOP_LEVEL_COUNT in the server row descriptor associated with *IDA*. *IDA* shall be one of exactly *TLC* foreign-data wrapper item descriptor areas in the server row 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 DECFLOAT, and PRECISION is a valid precision value for the DECFLOAT data type.

v) TYPE indicates CHARACTER, CHARACTER LARGE OBJECT, CHARACTER LARGE OBJECT LOCATOR, BINARY, BINARY VARYING, BINARY LARGE OBJECT, BINARY LARGE OBJECT LOCATOR, SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, USER-DEFINED TYPE LOCATOR, REF, or DATALINK.

vi) TYPE indicates ROW and, where *N* is the value of the DEGREE field, there are exactly *N* immediately subordinate descriptor areas of *IDA*, and those subordinate descriptor areas are valid.

vii) TYPE indicates ARRAY LOCATOR or MULTISET LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that subordinate descriptor area is valid.

viii) TYPE indicates an implementation-defined data type.

General Rules

None.

Conformance Rules

None.

21.2 Implicit foreign-data wrapper cursor

Function

Declare and open a foreign-data wrapper cursor.

General Rules

1) Let *AE* be an *ALLOCATED FDW-EXECUTION* specified in an application of this Subclause.
2) Let *SS* be the SQL-statement that is effectively associated with *AE*.
3) If there is no cursor effectively associated with *AE*, then:
   a) A cursor declaration descriptor *CDD* is created, as follows:
      i) The kind of cursor is a foreign-data wrapper cursor.
      ii) The provenance of the cursor is the SQL-session identifier of *AE*.
      iii) The name of the cursor is implementation-dependent.
      iv) The cursor's origin is *SS*.
      v) The cursor's declared properties are:
         1) The cursor's declared scrollability is 'NO SCROLL'.
         2) The cursor's declared sensitivity is 'ASENSITIVE'.
         3) The cursor's declared holdability is 'WITHOUT HOLD'.
         4) The cursor's declared returnability is 'WITHOUT RETURN'.
   b) A cursor instance descriptor *CID* is created, as follows:
      i) The cursor declaration descriptor is *CDD*.
      ii) The SQL-session identifier is the SQL-session identifier of *AE*.
      iii) The cursor's state is closed.
4) Cursor *CID* is effectively opened in the following steps:
   a) A copy *CS* 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, SYSTEM_USER, CURRENT_CATALOG, CURRENT_SCHEMA, CURRENT_PATH, CURRENT_DEFAULT_TRANSFORM_GROUP, or CURRENT_TRANSFORM_GROUP_FOR_TYPE <path-resolved user-defined type name> is effectively replaced by the value resulting from evaluation of CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, CURRENT_CATALOG, CURRENT_SCHEMA, CURRENT_PATH, CURRENT_DEFAULT_TRANSFORM_GROUP, or

CURRENT_TRANSFORM_GROUP_FOR_TYPE <path-resolved user-defined type name>, respectively, with all such evaluations effectively done at the same instant in time.

b) Let *QE* be the <query expression> simply contained in *CS*.

c) Let *T* be the table specified by *QE*.

d) A result set descriptor for *RSD* is effectively created as follows:

   i) The <cursor specification> is *CS*.

   ii) The sequence of rows is *T*.

   iii) The position is before the first row of *T*.

   iv) The operational properties are the same as the declared properties in *CD*.

e) Cursor *CN* is effectively placed in the open state with *RSD* as its result set descriptor.

Conformance Rules

None.

21.3 Implicit DESCRIBE INPUT USING clause

Function

Populate a specified descriptor area with information about the input values required to execute a foreign server request.

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 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 foreign server request and a numeric code that identifies the foreign server request, and are set to the value of the 'Identifier' and 'Code' columns, respectively, of the row in Table 36, "SQL-statement codes", that identifies in the 'SQL-statement' column the foreign server request.

4) A descriptor for the <dynamic parameter specification>s for the foreign server request is stored in *DESC* as follows:

   a) Let *D* be the number of <dynamic parameter specification>s in *S*. Let NS_i, 1 (one) ≤ *i* ≤ *D*, be the number of subordinate descriptors of the descriptor for the *i*-th input dynamic parameter.

   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 62 — 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) The implicitly ordered subordinate descriptors for the *j*-th <dynamic parameter specification>, if any, 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, DATA_POINTER, INDICATOR, OCTET_LENGTH, RETURNED_CARDINALITY, 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 [ISO9075-3], indicating the data type of the <dynamic parameter specification> or subordinate descriptor.

iii) NULLABLE is set to 1 (one).

NOTE 63 — 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 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; COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string's collation.

2) If TYPE indicates a <binary string type>, then LENGTH and OCTET LENGTH are both set to the length or maximum length in octets of the binary string.

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

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

5) 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 [ISO9075-3], to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision> if either is applicable.

6) 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 [ISO9075-3], 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.

7) 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.

8) 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>.

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

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

11) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.

Conformance Rules

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 foreign server request.

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 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 foreign server request and a numeric code that identifies the foreign server request, and are set to the value of the 'Identifier' and 'Code' columns, respectively, of the row in Table 36, "SQL-statement codes", that identifies in the 'SQL-statement' column the foreign server request.

4) A representation of the column descriptors of the <select list> columns for the foreign server request 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*. Let NS_i, 1 (one) ≤ *i* ≤ *D*, be the number of subordinate descriptors of the descriptor for the *i*-th column of *T*.

         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 column 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) The implicitly ordered subordinate descriptors for the *j*-th column, if any, 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, DATA_POINTER, INDICATOR, and OCTET_LENGTH 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 [ISO9075-3], 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 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; COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string's collation.

2) If TYPE indicates a <binary string type>, then LENGTH and OCTET LENGTH are both set to the length or maximum length in octets of the binary string.

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

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

5) 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 [ISO9075-3], to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision> if either is applicable.

6) 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 [ISO9075-3], 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.

7) 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.

8) 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>.

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

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

11) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.

Conformance Rules

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* and *AE* be a *TYPE* and *ALLOCATED FWD-EXECUTION* specified in the rules of this Subclause.

2) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

3) Let *WPD* and *SPD* be the wrapper parameter descriptor and server parameter descriptor, respectively, for *AE*.

4) *WPD* and *SPD* describe the <dynamic parameter specification>s and <dynamic parameter specification> values, respectively, for the foreign server request being executed. 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) Let *AD* be the minimum of *NSPD* and *NWPD*.

   c) For each of the first *AD* item descriptor areas of *SPD*, if TYPE indicates DEFAULT, then:

      i) Let *TP*, *P*, and *SC* be the values of the TYPE, PRECISION, and SCALE fields, respectively, for the corresponding item descriptor area of *WPD*.

      ii) 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.

   d) If the first *AD* item descriptor areas of *SPD* are not valid as specified in Subclause 21.1, "Description of foreign-data wrapper item descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

   e) 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 *NWPD*, 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 INDICATOR 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 INDICATOR is not negative.

B) TYPE indicates ARRAY, ARRAY LOCATOR, MULTISET, or MULTISET LOCATOR.

C) Case:

I) If *HL1* and *HL2* are both pointer-supporting languages, then 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.

II) Otherwise, the value of DATA is not a valid value of the data type represented by the item descriptor area.

f) 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) Case:

A) If *HL1* and *HL2* are both pointer-supporting languages, then let *V* be the value of the host variable addressed by DATA_POINTER.

B) Otherwise, let *V* be the value of DATA.

2) Case:

A) If TYPE indicates CHARACTER, then let *Q* be the value of OCTET_LENGTH 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.

g) Let *TDT* be the effective data type of the *i*-th 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.

h) If *SDT* is an array locator data type or multiset locator data type, then let *TV* be the value *SV*.

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

Case:

i) If *SDT* and *TDT* are binary string types, then the <cast specification>


```
CAST ( SV AS TDT )
```

is effectively performed and the result is the value *TV* of the *i*-th parameter.

ii) If *SDT* and *TDT* are numeric data types, then the <cast specification>

```
CAST ( SV AS TDT )
```

is effectively performed and the result is the value *TV* of the *i*-th parameter.

iii) Otherwise, the <cast specification>

```
CAST ( SV AS TDT )
```

is effectively performed and the result is the value *TV* of the *i*-th parameter.

Conformance Rules

None.

21.6 Implicit FETCH USING clause

Function

Specify the rules for an implicit FETCH USING clause.

General Rules

1) Let *OE* be an *OPENED FDW-EXECUTION* specified in the rules of this Subclause.

2) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

3) Case:

   a) If the PASSTHROUGH flag associated with *OE* is *True*, then let *RD* be the wrapper row descriptor associated with *OE*.

   b) Otherwise, let *RD* be the table reference descriptor associated with *OE*.

4) Let *SRD* be the server row descriptor associated with *OE*.

5) *RD* and *SRD* describe the <select list> columns and <target specification>s, respectively, for the column values that are to be retrieved.

   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) Case:

      i) If *HL1* and *HL2* are both pointer-supporting languages, then 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*.

      ii) Otherwise, 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> 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 foreign-data wrapper 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 *RD* whose LEVEL is 0 (zero) and all of its subordinate descriptor areas.

e) Let *TYPE*, *OL*, *D*, *DP*, *IP*, and *LP* be the values of the TYPE, OCTET_LENGTH, DATA, DATA_POINTER, INDICATOR, and OCTET_LENGTH 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).

f) Let *SV* be the value of the <select list> column, with data type *SDT*.

g) 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 [ISO9075-3].

    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 item descriptor area of *SRD* whose LEVEL is 0 (zero) and all of its subordinate descriptor areas.

i) If *TDT* is an array locator data type or a multiset locator data type, then

  Case:

  i) 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*.

  ii) Otherwise, the value *TV* of the *i*-th bound target is the null value.

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

  Case:

  i) If *SDT* and *TDT* are binary string types, then the <cast specification>

```
CAST ( SV AS TDT )
```

  is effectively performed and the result is the value *TV* of the *i*-th parameter.

  ii) If *SDT* and *TDT* are numeric data types, then the <cast specification>

```
CAST ( SV AS TDT )
```

  is effectively performed and the result is the value *TV* of the *i*-th parameter.

  iii) Otherwise, the <transcoding>

```
CONVERT ( CAST ( SV AS
CHARACTER VARYING (M) ) USING UTF16 )
```

is effectively performed, where *M* is the implementation-defined maximum length of a variable-length character string, and the result is the value *TV* of the *i*-th parameter.

k) Let *IDA* be the top-level item descriptor area corresponding to the *i*-th bound column.

l) Case:

i) If TYPE indicates ROW, then

Case:

1) If *TV* is the null value, then the value of *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, ARRAY LOCATOR, MULTISET, or MULTISET LOCATOR, is set to the appropriate 'Code' for SQL NULL DATA in Table 27, "Miscellaneous codes used in CLI", in [ISO9075-3] and the value of the host variable addressed by *DP* and the values of *D* 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 GR 5)l) 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 the value of *IP* is set to the appropriate 'Code' for SQL NULL DATA in Table 27, "Miscellaneous codes used in CLI", in [ISO9075-3], and the value of the host variable addressed by *DP* and the value of *D* and the value of *LP* are implementation-dependent.

2) Otherwise:

A) The value of *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) Case:

a) If *HL1* and *HL2* are both pointer-supporting languages, then the General Rules of Subclause 21.7, "Character string retrieval", are applied with *DP*, *TV*, *OL*, and *LP* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

b) Otherwise, the General Rules of Subclause 21.7, "Character string retrieval", are applied with *D*, *TV*, *OL*, and *LP*, as *TARGET*, *VALUE*,

TARGET OCTET LENGTH, and *RETURNED OCTET LENGTH*, respectively.

II) If TYPE indicates BINARY, BINARY VARYING, or BINARY LARGE OBJECT, then

Case:

1) If *HL1* and *HL2* are both pointer-supporting languages, then the General Rules of Subclause 21.8, "Binary string retrieval", are applied with *DP*, *TV*, *OL*, and *LP* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

2) Otherwise, the General Rules of Subclause 21.8, "Binary string retrieval", are applied with *D*, *TV*, *OL*, and *LP* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

III) If TYPE indicates ARRAY, ARRAY LOCATOR, MULTISET, or MULTISET LOCATOR, then the value of RETURNED_CARDINALITY is set to the cardinality of *TV*.

IV) Otherwise,

Case:

1) If *HL1* and *HL2* are both pointer-supporting languages, then the value of the host variable addressed by *DP* is set to *TV* and the value of *LP* is implementation-dependent.

2) Otherwise, the value of *D* is set to *TV* and the value of *LP* is implementation-dependent.

Conformance Rules

None.

21.7 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: *FDW-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*.

Conformance Rules

None.

21.8 Binary string retrieval

Function

Specify the rules for retrieving binary 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: *FDW-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*.

Conformance Rules

None.

21.9 Tables used with SQL/MED

The tables contained in this Subclause are used to specify the codes used by the various foreign-data wrapper interface routines.

Table 27 — Codes used for <table reference> types

<table reference> type	Code
TABLE_NAME	1

Table 28 — Codes used for <value expression> kinds

<value expression> kind	Code
COLUMN_NAME	1
CONSTANT	2
OPERATOR	3
PARAMETER	4

Table 29 — Codes used for foreign-data wrapper 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

Field	Code	Type
Implementation-defined diagnostics status field	< 0	Status

Table 30 — Codes used for foreign-data wrapper 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
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
KEY_MEMBER	1030	KEY_MEMBER	Item
KEY_TYPE	1029	KEY_TYPE	Header
LENGTH	1003	LENGTH	Item
LEVEL	1042	LEVEL	Item

Field	Code	SQL Item Descriptor Name	Type
NAME	1011	NAME	Item
NULLABLE	1008	NULLABLE	Item
OCTET_LENGTH	1013	OCTET_LENGTH	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_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

Field	Code	SQL Item Descriptor Name	Type
Implementation-defined foreign-data wrapper descriptor header field	0 (zero) through 999, or ≥ 1200, excluding values defined in this table	Implementation-defined foreign-data wrapper descriptor header field	Header
Implementation-defined foreign-data wrapper descriptor item field	0 (zero) through 999, or ≥ 1200, excluding values defined in this table	Implementation-defined foreign-data wrapper descriptor item field	Item

Table 31 — Codes used for foreign-data wrapper handle types

Handle type	Code
ExecutionHandle	1
FSConnectionHandle	2
ReplyHandle	3
RequestHandle	4
ServerHandle	6
TableReferenceHandle	7
UserHandle	8
ValueExpressionHandle	9
WrapperHandle	10
WrapperEnvHandle	11
DescriptorHandle	12

Table 32 — Ability to retrieve foreign-data wrapper 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	
INDICATOR		No		No
KEY_MEMBER	No		No	No
KEY_TYPE	No		No	No
LENGTH				
LEVEL				
NAME				

	May be retrieved			
Field	**SRD**	**WRD or TRD**	**SPD**	**WPD**
NULLABLE				
OCTET_LENGTH				
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_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				
USER_DEFINED_TYPE_SCHEMA				

	May be retrieved			
Field	**SRD**	**WRD or TRD**	**SPD**	**WPD**
Implementation-defined foreign-data wrapper descriptor header field	ID	ID	ID	ID
Implementation-defined foreign-data wrapper descriptor item field	ID	ID	ID	ID

† **Where** "No" means that the descriptor field is not retrievable, *PS* means that the descriptor field is retrievable from the IRD only when a prepared or executed statement is associated with the IRD, the absence of any notation means that the descriptor field is retrievable, and "ID" means that it is implementation-defined whether or not the descriptor field is retrievable.

Table 33 — Ability to set foreign-data wrapper descriptor fields

	May be set			
Field	**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	

	May be set			
Field	**SRD**	**WRD or TRD**	**SPD**	**WPD**
DYNAMIC_FUNCTION	No	No	No	No
DYNAMIC_FUNCTION_CODE	No	No	No	No
INDICATOR		No		No
KEY_MEMBER	No	No	No	No
KEY_TYPE	No	No	No	No
LENGTH		No		
LEVEL		No		
NAME		No		
NULLABLE		No		
OCTET_LENGTH		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_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

	May be set			
Field	**SRD**	**WRD or TRD**	**SPD**	**WPD**
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 foreign-data wrapper descriptor header field	ID	ID	ID	ID
Implementation-defined foreign-data wrapper descriptor item field	ID	ID	ID	ID

† **Where** "No" means that the descriptor field is not settable, "ID" means that it is implementation-defined whether or not the descriptor field is settable, and the absence of any notation means that the descriptor field is settable.

Table 34 — Foreign-data wrapper descriptor field default values

	Default values			
Field	**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)	

	Default values			
Field	**SRD**	**WRD or TRD**	**APD**	**WPD**
CURRENT_TRANSFORM_GROUP				
DATA				
DATA_POINTER	Null		Null	
DATETIME_INTERVAL_CODE				
DATETIME_INTERVAL_PRECISION				
DEGREE				
DYNAMIC_FUNCTION				
DYNAMIC_FUNCTION_CODE				
INDICATOR				
KEY_MEMBER				
KEY_TYPE				
LENGTH				
LEVEL	0 (zero)			
NAME				
NULLABLE				
OCTET_LENGTH				
PARAMETER_MODE				
PARAMETER_ORDINAL_POSITION				
PARAMETER_SPECIFIC_CATALOG				
PARAMETER_SPECIFIC_NAME				
PARAMETER_SPECIFIC_SCHEMA				
PRECISION				
RETURNED_CARDINALITY				

	Default values			
Field	**SRD**	**WRD or TRD**	**APD**	**WPD**
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 foreign-data wrapper descriptor header field	ID	ID	ID	ID
Implementation-defined foreign-data wrapper descriptor item field	ID	ID	ID	ID

† **Where** "Null" means that the descriptor field's default value is a null pointer, the absence of any notation means that the descriptor field's default value is initially undefined, "ID" means that the descriptor field's default value is implementation-defined, and any other value specifies the descriptor field's default value.

Table 35 — Codes used for the format of the character string transmitted by GetSQLString()

Format	Code
SQL-string format	1
Implementation-defined formats	*x*[1]

Format	Code
[1] An implementation-defined negative number different from the value associated with any other format.	

(Blank page)

22 Foreign-data wrapper interface routines

22.1 <foreign-data wrapper interface routine>

Function

Describe a generic foreign-data wrapper interface routine.

Format

```
<foreign-data wrapper interface routine> ::=
  <foreign-data wrapper interface routine prefix>
      <foreign-data wrapper interface routine generic>

<foreign-data wrapper interface routine prefix> ::=
  MED

<foreign-data wrapper interface routine generic> ::=
  <foreign-data wrapper interface routine name>
      <foreign-data wrapper parameter list>
      [ <foreign-data wrapper returns clause> ]

<foreign-data wrapper interface routine name> ::=
    AdvanceInitRequest
  | AllocDescriptor
  | AllocQueryContext
  | AllocWrapperEnv
  | Close
  | ConnectServer
  | FreeDescriptor
  | FreeExecutionHandle
  | FreeFSConnection
  | FreeQueryContext
  | FreeReplyHandle
  | FreeWrapperEnv
  | GetAuthorizationId
  | GetBoolVE
  | GetDescriptor
  | GetDiagnostics
  | GetDistinct
  | GetNextReply
  | GetNumBoolVE
  | GetNumChildren
  | GetNumOrderByElems
  | GetNumReplyBoolVE
  | GetNumReplyOrderBy
  | GetNumReplySelectElems
  | GetNumReplyTableRefs
  | GetNumRoutMapOpts
```

```
| GetNumSelectElems
| GetNumServerOpts
| GetNumTableColOpts
| GetNumTableOpts
| GetNumTableRefElems
| GetNumUserOpts
| GetNumWrapperOpts
| GetOpts
| GetOrderByElem
| GetReplyBoolVE
| GetReplyCardinality
| GetReplyDistinct
| GetReplyExecCost
| GetReplyFirstCost
| GetReplyOrderElem
| GetReplyReExecCost
| GetReplySelectElem
| GetReplyTableRef
| GetRoutineMapping
| GetRoutMapOpt
| GetRoutMapOptName
| GetSelectElem
| GetSelectElemType
| GetServerName
| GetServerOpt
| GetServerOptByName
| GetServerType
| GetServerVersion
| GetSPDHandle
| GetSQLString
| GetSRDHandle
| GetStatistics
| GetTableColOpt
| GetTableColOptByName
| GetTableOpt
| GetTableOptByName
| GetTableRefElem
| GetTableRefElemType
| GetTableRefTableName
| GetTableServerName
| GetTRDHandle
| GetUserOpt
| GetUserOptByName
| GetValExprColName
| GetValueExpDesc
| GetValueExpKind
| GetValueExpName
| GetValueExpTable
| GetVEChild
| GetWPDHandle
| GetWrapperLibraryName
| GetWrapperName
| GetWrapperOpt
| GetWrapperOptByName
| GetWRDHandle
| InitRequest
| Iterate
```



```
  | Open
  | ReOpen
  | SetDescriptor
  | TransmitRequest

<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 interface 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) A <foreign-data wrapper interface routine> defines a predefined routine written in a programming language that is invoked by a compilation unit of the same programming language. Let *HL* be that programming language. *HL* shall be one of Ada, C, COBOL, Fortran, M, Pascal, or PL/I.

2) A <foreign-data wrapper interface routine> that contains a <foreign-data wrapper returns clause> is called a *foreign-data wrapper interface function*. A <foreign-data wrapper interface routine> that does not contain a <foreign-data wrapper returns clause> is called a *foreign-data wrapper interface procedure*.

3) For each foreign-data wrapper interface function *WF*, there is a corresponding foreign-data wrapper interface procedure *WP*, with the same <foreign-data wrapper interface 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*.

   b) If <foreign-data wrapper parameter mode> is OUT, then the parameter is an *output parameter*.

c) If <foreign-data wrapper parameter mode> is INOUT, then the parameter is both an input parameter and an output parameter.

NOTE 64 — An output parameter is either a non-pointer host variable passed by reference or a pointer host variable passed by value.

6) There shall be no <separator> between the <foreign-data wrapper interface routine prefix> and the <foreign-data wrapper interface routine generic> in a <foreign-data wrapper interface routine name>.

7) Let *WR* be a <foreign-data wrapper interface routine> and let *RN* be its <foreign-data wrapper interface 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*.

8) Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 19.5, "SQL/CLI 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".

9) 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 interface routine is a foreign-data wrapper interface 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 interface routine> are specified in Subclause 22.2, "<foreign-data wrapper interface routine> invocation".

Conformance Rules

1) Without Feature M018, "Foreign-data wrapper interface routines in Ada", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in Ada.

2) Without Feature M019, "Foreign-data wrapper interface routines in C", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in C.

3) Without Feature M020, "Foreign-data wrapper interface routines in COBOL ", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in COBOL.

4) Without Feature M021, "Foreign-data wrapper interface routines in Fortran", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in Fortran.

5) Without Feature M022, "Foreign-data wrapper interface routines in MUMPS ", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in M.

6) Without Feature M023, "Foreign-data wrapper interface routines in Pascal", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in Pascal.

7) Without Feature M024, "Foreign-data wrapper interface routines in PL/I", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> written in PL/I.

8) Without Feature M018, "Foreign-data wrapper interface routines in Ada", a conforming SQLserver shall not contain an invocation of a <foreign-data wrapper interface routine> written in Ada.

9) Without Feature M019, "Foreign-data wrapper interface routines in C", a conforming SQLserver shall not contain an invocation of a <foreign-data wrapper interface routine> written in C.

10) Without Feature M020, "Foreign-data wrapper interface routines in COBOL ", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> written in COBOL.

11) Without Feature M021, "Foreign-data wrapper interface routines in Fortran", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> written in Fortran.

12) Without Feature M022, "Foreign-data wrapper interface routines in MUMPS ", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> written in M.

13) Without Feature M023, "Foreign-data wrapper interface routines in Pascal", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> written in Pascal.

14) Without Feature M024, "Foreign-data wrapper interface routines in PL/I", a conforming SQLserver shall not contain an invocation of a <foreign-data wrapper interface routine> written in PL/I.

22.2 <foreign-data wrapper interface routine> invocation

Function

Specify the rules for invocation of a <foreign-data wrapper interface routine>.

Syntax Rules

1) Let *HL* be the programming language of *CP*, the caller of a <foreign-data wrapper interface routine>.

2) A foreign-data wrapper interface function or foreign-data wrapper interface procedure is invoked by the *HL* mechanism for invoking functions or procedures, respectively.

3) Let *RN* be the <foreign-data wrapper interface routine name> of the <foreign-data wrapper interface 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 interface routine>".

5) Each argument to a <foreign-data wrapper interface routine> that is of type CHARACTER(*n*) shall be passed by reference, according to the mechanisms of *HL*.

   Case:

   a) Of *HL* is C, then each input argument to a <foreign-data wrapper interface routine> that is not of type CHARACTER(*n*) shall be passed by value. Each output argument to a <foreign-data wrapper interface routine> that is not of type CHARACTER(*n*) that identifies a non-pointer host variable shall be passed by reference; each output argument to a <foreign-data wrapper interface routine> that is not of type CHARACTER(*n*) that identifies a pointer host variable shall be passed by value.

   b) Otherwise, each input or output argument to a <foreign-data wrapper interface routine> that is not of type CHARACTER(*n*) shall be passed by reference.

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 interface 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 interface 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 interface routine> is a foreign-data wrapper interface function, then:

      i) The values of all output arguments are established.

      ii) Let *RC* be the return value.

   b) If the <foreign-data wrapper interface routine> is a foreign-data wrapper interface 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: *warning*, then *RC* is set to indicate **Success with information**.

         3) 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: *FDW-specific condition — invalid handle*, then *RC* is set to indicate **Invalid handle**.

         2) Otherwise, *RC* is set to indicate **Error**.

      iv) If *RN* is a foreign-data wrapper interface wrapper routine, then the actions of invoking the SQL-server in response to the failed execution of *RN* are implementation-dependent.

Conformance Rules

None.

22.3 Foreign-data wrapper interface wrapper routines

22.3.1 AdvanceInitRequest

Function

Determine whether a foreign-data wrapper can execute a foreign server request and gather multiple different FDW-replies and FDW-executions.

Definition

```
AdvanceInitRequest (
    FSConnectionHandle    IN      INTEGER,
    RequestHandle         IN      INTEGER,
    ReplyHandle           OUT     INTEGER,
    ExecutionHandle       OUT     INTEGER,
    QueryContextHandle    IN      INTEGER )
RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle.
2) If *FSCH* does not identify an allocated FSconnection, then an exception condition is raised: *FDW-specific condition — invalid handle*.
3) Let *QCH* be the value of QueryContextHandle.
4) If *QCH* does not identify an allocated query context, then an exception condition is raised: *FDW-specific condition — invalid handle*.
5) Let *RQH* be the value of RequestHandle.
6) A set *SRH* of pairs of reply handles and corresponding execution handles is created as follows:
   a) Let RPH_i and EXH_i be the ReplyHandle and ExecutionHandle, respectively, that would be returned by an invocation of `InitRequest()` with *FSCH* and *RQH* as input arguments.
   b) RPH_i and EXH_i are the *i*-th pair included in *SRH*.
7) Let *RPH* and *EXH* be a pair of reply handle and execution handle included in *SRH*, chosen in a foreign-data wrapper implementation-dependent way.
8) ReplyHandle is set to *RPH*.
9) ExecutionHandle is set to *EXH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains AdvanceInitRequest.

22.3.2 AllocQueryContext

Function

Allocate a query context and assign a handle to it.

Definition

```
AllocQueryContext (
    FSConnectionHandle    IN      INTEGER,
    QueryContextHandle    OUT     INTEGER )
RETURNS SMALLINT
```

General Rules

1) Let *FSCH* be the value of FSConnectionHandle.

2) If *FSCH* does not identify an allocated FSConnection, then an exception condition is raised: *FDW-specific condition — invalid handle*.

3) If the foreign-data wrapper implementation-dependent maximum number of query contexts that can be allocated at one time has already been reached, then an exception condition is raised: *FDW-specific condition — limit on number of handles exceeded*. A skeleton query context is allocated and is assigned a unique value that is returned in QueryContextHandle.

4) Case:

   a) If the memory requirements to manage a query context cannot be satisfied, then QueryContextHandle is set to 0 (zero) and an exception condition is raised: *FDW-specific condition — memory allocation error*.

      NOTE 65 — No diagnostic information is generated in this case, as there is no valid QueryContextHandle that can be used to obtain diagnostics information.

   b) If the resources to manage a query context cannot be allocated for foreign-data wrapper implementation-defined reasons, then an implementation-defined exception condition is raised. A skeleton query context is allocated and is assigned a unique value that is returned in QueryContextHandle.

   c) Otherwise, the resources to manage a query context are allocated and are referred to as an *allocated query context*. The allocated query context is assigned a unique value that is returned in QueryContextHandle.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains AllocQueryContext.

22.3.3 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: *FDW-specific condition — invalid handle*.

3) If the implementation-defined maximum number of foreign-data wrapper environments that can be allocated at one time has already been reached, then an exception condition is raised: *FDW-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: *FDW-specific condition — memory allocation error*.

      NOTE 66 — 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 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.

6) Let *WN* be the WrapperName that would be returned by an invocation of `GetWrapperName()` with *WH* as the WrapperHandle parameter.

7) Let *WL* be the WrapperLibraryName that would be returned by an invocation of `GetWrapperLibrary-Name()` with *WH* as the WrapperHandle parameter.

8) It is implementation-dependent what use the `AllocWrapperEnv()` routine makes of the values of *WN* and *WL*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains AllocWrapperEnv.

22.3.4 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: *FDW-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 result set descriptor 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

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains Close.

22.3.5 ConnectServer

Function

Establish a connection to a foreign server and assign a handle to it.

Definition

```
ConnectServer (
    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: *FDW-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: *FDW-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: *FDW-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 foreign-data wrapper 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 `GetAuthorizationId()` with *UH* as the UserHandle parameter.

8) Let *SN* be the ServerName that would be returned by an invocation of `GetServerName()` with *SH* as the ServerHandle parameter.

9) Let *ST* be the ServerType that would be returned by an invocation of `GetServerType()` with *SH* as the ServerHandle parameter.

10) Let *SV* be the ServerVersion that would be returned by an invocation of `GetServerVersion()` with *SH* as the ServerHandle parameter.

11) Let *E* be the FDW-environment identified by WrapperEnvHandle.

12) The foreign-data wrapper diagnostics area associated with *E* is emptied.

13) If the implementation-defined 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: *FDW-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: *FDW-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: *FDW-specific condition — unable to establish connection*.

    b) Otherwise, the connection to *FS* is established.

16) It is implementation-dependent what use the foreign-data wrapper makes of the values of *UN*, *SN*, *ST*, and *SV*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains ConnectServer.

22.3.6 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: *FDW-specific condition — invalid handle*.

3) Let *E* be the FDW-execution identified by *EH*.

4) The foreign-data wrapper diagnostics area associated with *E* is emptied.

5) If there is a foreign-data wrapper cursor *CR* associated with *E*, then:

   a) If *CR* is open, then:

      i) *CR* is placed in the closed state and its result set descriptor is destroyed.

      ii) Any fetched row associated with *E* is removed from association with *E*.

   b) The cursor declaration descriptor and cursor instance descriptor of *CR* are destroyed.

6) Case:

   a) If the PASSTHROUGH flag associated with *EH* is *False*, then:

      i) Let *SRD* be the server row descriptor associated with *E* and let *SRDHandle* be the descriptor handle that identifies *SRD*. The `FreeDescriptor()` routine is invoked with *SRDHandle* as the DescriptorHandle parameter.

      ii) Let *SPD* be the server parameter descriptor associated with *E* and let *SPDHandle* be the descriptor handle that identifies *SPD*. The `FreeDescriptor()` routine is invoked with *SPDHandle* as the DescriptorHandle parameter.

   b) Otherwise:

      i) Let *SRD* be the server row descriptor associated with *E* and let *SRDHandle* be the descriptor handle that identifies *SRD*. The `FreeDescriptor()` routine is invoked with *SRDHandle* as the DescriptorHandle parameter.

ii) Let *SPD* be the server parameter descriptor associated with *E* and let *SPDHandle* be the descriptor handle that identifies *SPD*. The `FreeDescriptor()` routine is invoked with *SPDHandle* as the DescriptorHandle parameter.

iii) Let *WRD* be the wrapper row descriptor associated with *E* and let *WRDHandle* be the descriptor handle that identifies *WRD*. The `FreeDescriptor()` routine is invoked with *WRDHandle* as the DescriptorHandle parameter.

iv) Let *WPD* be the wrapper parameter descriptor associated with *E* and let *WPDHandle* be the descriptor handle that identifies *WPD*. The `FreeDescriptor()` routine is invoked with *WPDHandle* as the DescriptorHandle parameter.

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

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains FreeExecutionHandle.

22.3.7 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: *FDW-specific condition — invalid handle*.

3) Let *C* be the allocated FS-connection identified by *FSCH*.

4) The foreign-data wrapper diagnostics area associated with *C* is emptied.

5) If an allocated query context is associated with *C*, then an exception condition is raised: *FDW-specific condition — function sequence error*.

6) *C* is deallocated and all its resources are freed.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains FreeFSConnection.

22.3.8 FreeQueryContext

Function

Deallocate a query context.

Definition

```
FreeQueryContext (
    QueryContextHandle    IN      INTEGER )
RETURNS SMALLINT
```

General Rules

1) Let *QCH* be the value of QueryContextHandle.

2) If *QCH* does not identify an allocated query context, then an exception condition is raised: *FDW-specific condition — invalid handle*.

3) Let *Q* be the allocated query context identified by *QCH*.

4) The foreign-data wrapper diagnostics area associated with *Q* is emptied.

5) If an allocated reply description or FDW-execution is associated with *Q*, then an exception condition is raised: *FDW-specific condition — function sequence error*.

6) *Q* is deallocated and all its resources are freed.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains FreeQueryContext.

22.3.9 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: *FDW-specific condition — invalid handle*.

3) Let *R* be the FDW-reply identified by *RH*.

4) The foreign-data wrapper diagnostics area associated with *R* is emptied.

5) *R* is deallocated and all its resources are freed.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains FreeReplyHandle.

22.3.10 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: *FDW-specific condition — invalid handle*.

3) Let *E* be the allocated FDW-environment identified by *WEH*.

4) The foreign-data wrapper diagnostics area associated with *E* is emptied.

5) If an allocated FS-connection is associated with *E*, then an exception condition is raised: *FDW-specific condition — function sequence error*.

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

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains FreeWrapperEnv.

22.3.11 GetNextReply

Function

Get a new reply handle and execution handle for a foreign server request.

Definition

```
GetNextReply (
    ReplyHandle           IN      INTEGER,
    NextReplyHandle       OUT     INTEGER,
    NextExecutionHandle   OUT     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: *FDW-specific condition — invalid handle*.

3) Let *SRH* be the set of reply handles that was allocated during the execution of the `AdvanceInitRequest()` routine during which *RH* was allocated.

4) Let *NRH* be a handle referencing an allocated reply description included in *SRH*.

5) Let *NEH* be the handle referencing an FDW-execution that corresponds to *NRH*.

6) NextReplyHandle is set to *NRH*.

7) NextExecutionHandle is set to *NEH*.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNextReply.

22.3.12 GetNumReplyBoolVE

Function

Get the number of <boolean value expression>s simply contained in the <where clause> of a query that the foreign-data wrapper is capable of handling.

Definition

```
GetNumReplyBoolVE (
    ReplyHandle           IN      INTEGER,
    NumberOfBoolVEs       OUT     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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <boolean value expression> elements simply contained in the <where clause> of *Q* that the foreign-data wrapper is capable of handling.

5) NumberOfBoolVEs is set to *N*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumReplyBoolVE.

22.3.13 GetNumReplyOrderBy

Function

Get the number of columns that are used to order the result that the foreign-data wrapper is capable of handling.

Definition

```
GetNumReplyOrderBy (
    ReplyHandle           IN      INTEGER,
    NumberOfOrderByElems  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: *FDW-specific condition — invalid handle*.

3) Let *NOE* be the number of <value expression>s used to order the result of the query associated with *RH* that the foreign-data wrapper is capable of handling.

4) NumberOfOrderByElems is set to *NOE*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumReplyOrderBy.

22.3.14 GetNumReplySelectElems

Function

Get the number of <value expression>s in the <select list> of a query that the foreign-data wrapper is capable of handling.

Definition

```
GetNumReplySelectElems (
    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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <value expression> elements simply contained in the <select list> of *Q* that the foreign-data wrapper is capable of handling.

5) NumberOfSelectListElements is set to *N*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumReplySelectElems.

22.3.15 GetNumReplyTableRefs

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

```
GetNumReplyTableRefs (
    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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) Let *N* be the number of <table reference> elements simply contained in the <from clause> of *Q* that the foreign-data wrapper is capable of handling.

5) NumberOfTableReferences is set to *N*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumReplyTableRefs.

22.3.16 GetOpts

Function

Request the foreign-data wrapper to supply information about the capabilities of a given object, and other information pertaining to that object.

Definition

```
GetOpts (
    InputHandle         IN      INTEGER,
    HandleType          IN      SMALLINT,
    ReturnFormat        OUT     INTEGER,
    Options             OUT     CHARACTER VARYING(L2),
    BufferLength        IN      INTEGER,
    StringLength        OUT     INTEGER )
RETURNS SMALLINT
```

where: *L2* is determined by the value of StringLength 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 31, "Codes used for foreign-data wrapper handle types", then an exception condition is raised: *FDW-specific exception — invalid handle*.

3) If *IH* does not identify a handle of the type indicated by *HT*, then an exception condition is raised: *FDW-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, a description *CD* of the capabilities of *FDW* is created.

   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, a description *CD* of the capabilities of *FS* is created.

      If *CD* is an XML document, then it shall be a valid XML document according to the following DTD:

```
<?xml version="1.0" encoding="charencoding" ?>
<!- SQL/MED GetOpts Document ->
<!- UTF-8 and UTF-16 are the only required encodings ->
<!ELEMENT SQLMEDGenericOptions (SQLMEDGenericOption)+ >
  <!ELEMENT SQLMEDGenericOption (#PCDATA)>
    <!ATTLIST SQLMEDGenericOption SQLMEDOptionName CDATA #REQUIRED>
    <!ATTLIST SQLMEDGenericOption
                SQLMEDOptionType (INTEGER | CHARACTER) #REQUIRED>
```

where *charencoding* is either UTF-8 or UTF-16.

NOTE 67 — The CDATA value of the SQLMEDOptionName attribute and the PCDATA text of the SQLMEDGenericOption tag are implementation-defined.

NOTE 68 — The DTD can be internal to the XML document or it can be an external DTD referenced by a URI as specified in the XML specification. The way in which the foreign-data wrapper knows the URI to specify in the XML document is implementation-defined.

5) The General Rules of Subclause 21.7, "Character string retrieval", are applied to Options, *CD*, *LO*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

6) Case:

   a) If *CD* is an XML document, then the value of ReturnFormat is set to one (1).

   b) If *CD* is in a format defined by the foreign-data wrapper, then the value of ReturnFormat is set to a value, defined by the foreign-data wrapper, that corresponds to that format.

NOTE 69 — All negative values are reserved for use by foreign-data wrappers. All non-negative values are reserved for use by this International Standard.

Conformance Rules

1) Without Feature M009, "GetOpts and GetStatistics routinesGetOpts and GetStatistics routines", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetOpts.

22.3.17 GetReplyBoolVE

Function

Get the ordinal position, within the <where clause> of a query, of a <boolean value expression> element that the foreign-data wrapper is capable of handling.

Definition

```
GetReplyBoolVE (
    ReplyHandle           IN      INTEGER,
    Index                 IN      SMALLINT,
    BoolVENumber          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: *FDW-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: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*. Let *WCQ* be the <where clause> of *Q*.

6) Let *N* be the number of <boolean value expression> elements simply contained in *WCQ* that the foreign-data wrapper is capable of handling. Let *BVEH* be a list containing only those *N* <boolean value expression> elements, in the same relative positions in which they appear in *WCQ*.

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) BoolVENumber is set to the ordinal position in *WCQ* of the <boolean value expression> element that is *I*-th within *BVEH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyBoolVE.

22.3.18 GetReplyCardinality

Function

Get an estimate of the cardinality of the result set associated with a given reply.

Definition

```
GetReplyCardinality (
    ReplyHandle           IN      INTEGER,
    ReplyCardinality      OUT     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: *FDW-specific condition — invalid handle*.

3) Let *C* be the estimated cardinality of the result set associated with *RH*.

   NOTE 70 — If the foreign-data wrapper has no means to estimate the cardinality of the result set associated with *RH*, then *C* is a foreign-data wrapper implementation-dependent default value.

4) ReplyCardinality is set to *C*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyCardinality.

22.3.19 GetReplyDistinct

Function

Get information as to whether the foreign-data wrapper is capable of providing distinct rows in the result set.

Definition

```
GetReplyDistinct (
    ReplyHandle           IN      INTEGER,
    IsDistinct            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: *FDW-specific condition — invalid handle*.

3) Case:

   a) If the foreign-data wrapper is capable of providing distinct rows in the result set, then IsDistinct is set to 1 (one).

   b) Otherwise, IsDistinct is set to 0 (zero).

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyDistinct.

22.3.20 GetReplyExecCost

Function

Get a value that represents the estimated "cost" to retrieve the result set associated with the reply. Larger values represent greater costs.

Definition

```
GetReplyExecCost (
    ReplyHandle           IN      INTEGER,
    ReplyTotalExecCost    OUT     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: *FDW-specific condition — invalid handle*.

3) Let *C* be the estimated cost to retrieve the result set associated with *RH*.

   NOTE 71 — If the foreign-data wrapper has no means to estimate the cost to retrieve the result set associated with *RH*, then *C* is a foreign-data wrapper implementation-dependent default value.

4) ReplyTotalExecCost is set to *C*.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyExecCost.

22.3.21 GetReplyFirstCost

Function

Get a value that represents the estimated "cost" to retrieve the first row of the result set associated with the reply. Larger values represent greater costs.

Definition

```
GetReplyFirstCost (
    ReplyHandle           IN      INTEGER,
    ReplyExecFirstCost    OUT     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: *FDW-specific condition — invalid handle*.

3) Let *C* be the estimated cost to retrieve the first row of the result set associated with *RH*.

   NOTE 72 — If the foreign-data wrapper has no means to estimate the cost to retrieve the first row of the result set associated with *RH*, then *C* is a foreign-data wrapper implementation-dependent default value.

4) ReplyExecFirstCost is set to *C*.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyFirstCost.

22.3.22 GetReplyOrderElem

Function

Get the ordinal position, within the <select list>, of a <value expression> element that the foreign-data wrapper is capable of handling and that is used to order the result of a query.

Definition

```
GetReplyOrderElem (
    ReplyHandle           IN      INTEGER,
    Index                 IN      SMALLINT,
    OrderByNumber         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: *FDW-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: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*. Let *SLQ* be the <select list> of *Q*.

6) Let *N* be the number of <value expression> elements simply contained in *SLQ* that the foreign-data wrapper is capable of handling. Let *VEH* be a list containing only those *N* <value expression> elements, in the same relative positions in which they appear in *SLQ*.

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) OrderByNumber is set to the ordinal position in *SLQ* of the <value expression> element that is *I*-th within *VEH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyOrderElem.

22.3.23 GetReplyReExecCost

Function

Get a value that represents the estimated "cost" to re-execute the reply identified by the provided reply handle. Larger values represent greater costs.

Definition

```
GetReplyReExecCost (
    ReplyHandle           IN      INTEGER,
    ReplyReExecutionCost  OUT     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: *FDW-specific condition — invalid handle*.

3) Let *C* be the estimated cost to re-execute the reply identified by *RH*.

   NOTE 73 — If the foreign-data wrapper has no means to estimate the cost to re-execute *RH*, the *C* is a foreign-data wrapper implementation-dependent default value.

4) ReExecutionCost is set to *C*.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyReExecCost.

22.3.24 GetReplySelectElem

Function

Get the ordinal position, within the <select list> of a query, of a <value expression> element that the foreign-data wrapper is capable of handling.

Definition

```
GetReplySelectElem (
    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: *FDW-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: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*. Let *SLQ* be the <select list> of *Q*.

6) Let *N* be the number of <value expression>s simply contained in *SLQ* that the foreign-data wrapper is capable of handling. Let *VEH* be a list containing only those *N* <value expression> elements, in the same relative positions in which they appear in *SLQ*.

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) SelectListElementNumber is set to the ordinal position in *SLQ* of the <value expression> element that is *I*-th within *VEH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplySelectElem.

22.3.25 GetReplyTableRef

Function

Get the ordinal position, within the <from clause> of a query, of a <table reference> element that the foreign-data wrapper is capable of handling.

Definition

```
GetReplyTableRef (
    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: *FDW-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: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*. Let *FCQ* be the <from clause> of *Q*.

6) Let *N* be the number of <table reference>s simply contained in *FCQ* that the foreign-data wrapper is capable of handling. Let *TRH* be a list containing only those *N* <table reference>s, in the same relative positions in which they appear in *FCQ*.

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) TableReferenceNumber is set to the ordinal position in *FCQ* of the <table reference> element that is *I*-th within *TRH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetReplyTableRef.

22.3.26 GetSPDHandle

Function

Get the descriptor handle of the server parameter descriptor associated with a given 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: *FDW-specific condition — invalid handle*.

3) SPDHandle is set to the descriptor handle of the server parameter descriptor associated with *EH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetSPDHandle.

22.3.27 GetSRDHandle

Function

Get the descriptor handle of the server row descriptor associated with a given execution handle.

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: *FDW-specific condition — invalid handle*.

3) SRDHandle is set to the descriptor handle of the server row descriptor associated with *EH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetSRDHandle.

22.3.28 GetStatistics

Function

Retrieve implementation-defined statistics associated with a foreign server request.

Definition

```
GetStatistics (
    ExecutionHandle     IN      INTEGER,
    ReturnFormat        OUT     INTEGER,
    Statistics          OUT     CHARACTER VARYING(L),
    BufferLength        IN      INTEGER,
    StringLength        OUT     INTEGER )
RETURNS SMALLINT
```

where: *L* is equal to the value of StringLength 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: *FDW-specific condition — invalid handle*.

3) Case:

   a) If the foreign-data wrapper is able to report statistics associated with the foreign server request associated with *EH*, then a report of those statistics is created. If the report is in the form of an XML document, then it shall be a valid XML document according to the following DTD.

```
<?xml version="1.0" encoding="charencoding" ?>
<!– SQL/MED GetStatistics Document –>
<!– UTF-8 and UTF-16 are the only required encodings –>
<!ELEMENT SQLMEDStatisticsSet (SQLMEDStatistics)+ >
  <!ELEMENT SQLMEDStatistics (#PCDATA)>
    <!ATTLIST SQLMEDStatistics SQLMEDStatisticName CDATA #REQUIRED>
    <!ATTLIST SQLMEDStatistics
                SQLMEDStatisticType (INTEGER | CHARACTER) #REQUIRED>
```

   where *charencoding* is either UTF-8 or UTF-16.

   NOTE 74 — The CDATA values of the SQLMEDStatisticName attribute and the PCDATA text of the SQLMEDStatistics tag are implementation-defined.

   NOTE 75 — The DTD can be internal to the XML document or it can be an external DTD referenced by a URI as specified in the XML specification. The way in which the foreign-data wrapper knows the URI to specify in the XML document is implementation-defined.

   b) Otherwise, a completion condition is raised: *no data*.

4) The General Rules of Subclause 21.7, "Character string retrieval", are applied to Statistics, *SI*, *LOS*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

5) Case:

   a) If *SI* is an XML document, then the value of ReturnFormat is set to one (1).

   b) If *SI* is in a format defined by the foreign-data wrapper, then the value of ReturnFormat is set to a value defined by the foreign-data wrapper that corresponds to that format.

      NOTE 76 — All negative values are reserved for use by foreign-data wrappers. All non-negative values are reserved for use by this part of ISO/IEC 9075.

Conformance Rules

1) Without Feature M009, "GetOpts and GetStatistics routinesGetOpts and GetStatistics routines", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetStatistics.

22.3.29 GetWPDHandle

Function

Get the descriptor handle of the wrapper parameter descriptor associated with a given execution handle.

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: *FDW-specific condition — invalid handle*.

3) WPDHandle is set to the descriptor handle of the wrapper parameter descriptor associated with *EH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetWPDHandle.

22.3.30 GetWRDHandle

Function

Get the descriptor handle of the wrapper row descriptor associated with a given execution handle.

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: *FDW-specific condition — invalid handle*.

3) WRDHandle is set to the descriptor handle of the wrapper row descriptor associated with *EH*.

Conformance Rules

1) Without Feature M010, “Foreign-data wrapper support”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetWRDHandle.

22.3.31 InitRequest

Function

Determine whether a foreign-data wrapper can execute a given foreign server request.

Definition

```
InitRequest (
    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 FSconnection, then an exception condition is raised: *FDW-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 foreign-data wrapper 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 *IG* be the indication of whether `GetSQLString()` will be invoked or not. It is foreign-data wrapper implementation-dependent whether *IG* is *True* or *False*.

   NOTE 77 — The only possible values for *IG* are *True* and *False*.

6) Case:

   a) If *IG* is *True*, then let *SS* be the SQLString value returned by an invocation of `GetSQLString()` with *RH* as the RequestHandle parameter.

   b) Otherwise:

      i) Let *NTRE* be the NumberOfTableReferenceElement values that would be returned by an invocation of `GetNumTableRefElems()` with *RH* as the RequestHandle parameter.

      ii) For 1 (one) ≤ *i* ≤ *NTRE*:

         1) Let TRH_i be the TableReferenceHandle that would be returned by invocation of `GetTableRefElem()` with *RH* as the RequestHandle parameter and *i* as the TableReferenceElementNumber parameter.

         2) Let $TRDH_i$ be the TableReferenceDescriptorHandle that would be returned by invocation of `GetTRDHandle()` with TRH_i as the TableReferenceHandle parameter.

3) Let NC_i be the value of the COUNT descriptor field that would be returned by invocation of `GetDescriptor()` with $TRDH_i$ as the DescriptorHandle parameter, 0 (zero) as the RecordNumber parameter, and the code for COUNT from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

4) For 1 (one) ≤ j ≤ NC_i, let DT_{ij} be the effective data type of the j-th column, as represented by the values of the TYPE, LENGTH, OCTET_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 be returned by separate invocations of `GetDescriptor()` 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

5) Let TRT_i be the TableReferenceType that would be returned by an invocation of `GetTableRefElemType()` with TRH_i as the TableReferenceHandle parameter.

6) Let TN_i be the TableName that would be returned by invocation of `GetTableRefTableName()` with TRH_i as the TableReferenceHandle parameter.

iii) Let *NSLE* be the NumberOfSelectListElements that would be returned by an invocation of `GetNumSelectElems()` with *RH* as the RequestHandle parameter.

iv) For 1 (one) ≤ k ≤ *NSLE*, let VEH_k be the ValueExpressionHandle that would be returned by invocation of `GetSelectElem()` with *RH* as the RequestHandle parameter and k as the SelectListElementNumber parameter.

v) Let *NBVE* be the NumberOfBoolVE values that would be returned by invocation of `GetNumBoolVE()` with *RH* as the RequestHandle parameter.

vi) For 1 (one) ≤ k ≤ *NBVE*, let VEH_{k+NSLE} be the ValueExpressionHandle that would be returned by an invocation of `GetBoolVE()` with *RH* as the RequestHandle parameter, and k as the BoolVENumber parameter.

vii) For 1 (one) ≤ m ≤ *NSLE*+*NBVE*, let VET_m be the ValueExpressionKind that would be returned by an invocation of `GetValueExpKind()` with VEH_m as the ValueExpressionHandle parameter, and let CN_m be the ColumnName that would be returned by invocation of `GetValExprColName()` with VEH_m as the ValueExpressionHandle parameter.

7) If the implementation-defined 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: *FDW-specific condition — limit on number of handles exceeded*.

8) 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: *FDW-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.

9) Case:

a) If *IG* is *False* and, for any reason, the foreign-data wrapper cannot create an FDW-reply that corresponds to *RH* as described by *NTRE*, (TRH_i, $TRDH_i$, NC_i, TRT_i, and TN_i, for 1 (one) ≤ i ≤ *NTRE*), (DT_{ij}, for 1 (one) ≤ i ≤ *NTRE* and 1 (one) ≤ j ≤ NC_i), *NSLE*, and (VEH_k, VET_k, and CN_k, for 1 (one) ≤ k ≤ *NSLE+NBVE*), then an exception condition is raised: *FDW-specific condition — unable to create reply*.

NOTE 78 — One reason for raising this exception could be an Access Rule violation at the foreign server.

b) If *IG* is *True* and, for any reason, the foreign-data wrapper cannot create an FDW-reply that corresponds to *RH* as described by *SS*, then an exception condition is raised: *FDW-specific condition — unable to create reply*.

c) Otherwise, the FDW-reply corresponding to *RH* is created.

10) If the implementation-defined 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: *FDW-specific condition — limit on number of handles exceeded*.

11) 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: *FDW-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.

12) Case:

a) If *IG* is *False* and 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) ≤ i ≤ *NTRE*), (DT_{ij}, for 1 (one) ≤ i ≤ *NTRE* and 1 (one) ≤ j ≤ NC_i), *NSLE*, and (VEH_k, VET_k, and CN_k, for 1 (one) ≤ k ≤ *NSLE+NBVE*), then an exception condition is raised: *FDW-specific condition — unable to create execution*.

b) If *IG* is *False* and the foreign-data wrapper cannot create an FDW-execution that corresponds to *RH* as described by *SS*, then an exception condition is raised: *FDW-specific condition — unable to create execution*.

c) Otherwise, the FDW-execution corresponding to *RH* is created.

13) The PASSTHROUGH flag associated with the allocated FDW-execution is set to *False*.

14) Let *NIDA* be the number of item descriptor areas that are set up for the server row descriptor. Let *SRDHandle* be the DescriptorHandle that is returned by an invocation of the `AllocDescriptor()` routine with *NIDA* as the MaxDetailAreas parameter. Let *SRD* be the server row descriptor identified by *SRDHandle*. *SRD* is associated with the allocated FDW-execution.

    For this descriptor area, fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", are set to the specified default values by the invocation of the `SetDescriptor()` routine with *SRDHandle* as the DescriptorHandle parameter and *r* as the RecordNumber parameter, 1 (one) ≤ *r* ≤ *NIDA*, and to the codes for the fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. All other fields in the item descriptor areas of *SRD* are initially undefined.

15) Let *NIDAP* be the number of item descriptor areas that are set up for the server parameter descriptor. Let *SPDHandle* be the DescriptorHandle that is returned by an invocation of the `AllocDescriptor()` routine with *NIDAP* as the MaxDetailAreas parameter. Let *SPD* be the server parameter descriptor identified by *SPDHandle*. *SPD* is associated with the allocated FDW-execution.

    For this descriptor area, fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", are set to the specified default values by the invocation of the `SetDescriptor()` routine with *SPDHandle* as the DescriptorHandle parameter and *r* as the RecordNumber parameter, 1 (one) ≤ *r* ≤ *NIDAP*, and to the codes for the fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. All other fields in the item descriptor areas of *SPD* are initially undefined.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains InitRequest.

22.3.32 Iterate

Function

Retrieve the next row from an 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: *FDW-specific condition — function sequence error*.

3) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

4) Let *S* be the opened FDW-execution identified by ExecutionHandle.

5) Let *CR* be the open foreign-data wrapper cursor effectively associated with *S* and let *T* be the sequence of rows included in the result set descriptor of *CR*.

6) Let *SRD* be the server row descriptor for *S* and let *N* be the value of the TOP_LEVEL_COUNT field of *SRD*.

7) Case:

   a) If *HL1* and *HL2* are both pointer-supporting languages, then 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*.

   b) Otherwise, 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> as a *bound target* and refer to the corresponding <select list> column as a *bound column*.

8) Let *BC* be the number of the bound targets.

9) For *i*, 1 (one) ≤ *i* ≤ *BC*:

   a) 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*.

   b) If *TT* indicates DEFAULT, then:

      i) Case:

1) If the PASSTHROUGH flag associated with *EH* is *True*, then let *RD* be the wrapper row descriptor associated with *S*.

2) Otherwise, let *RD* be the table reference descriptor associated with *S*.

ii) Let *CT*, *P*, and *SC* be the values of the TYPE, PRECISION, and SCALE fields, respectively, for the item descriptor area of *RD* corresponding to the *i*-th bound column.

iii) 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.

10) 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 values are assigned to bound targets.

c) A completion condition is raised: *no data* and no further rules of this Subclause are applied.

11) 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*.

12) *NR* becomes the current row of *CR*.

13) 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 21.6, "Implicit FETCH USING clause", are applied with *S* as *OPENED FDW-EXECUTION*.

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

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains Iterate.

22.3.33 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: *FDW-specific condition — invalid handle*.

3) If *EH* identifies an opened FDW-execution, then an exception condition is raised: *FDW-specific condition — function sequence error*.

4) Let *S* be the allocated FDW-execution identified by ExecutionHandle.

5) If the PASSTHROUGH flag associated with *EH* is *True*, then:

   a) Let *SRD* be the SRDHandle that would be returned by an invocation of the `GetSRDHandle()` routine with *EH* as the ExecutionHandle parameter. Let *SPD* be the SPDHandle that would be returned by an invocation of the `GetSPDHandle()` routine with *EH* as the ExecutionHandle parameter. Let *WRD* be the WRDHandle that would be returned by an invocation of the `GetWRDHandle()` routine with *EH* as the ExecutionHandle parameter. Let *WPD* be the WPDHandle that would be returned by an invocation of the `GetWPDHandle()` routine with *EH* as the ExecutionHandle parameter.

   b) Let *NCR* 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 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

   c) Let *DT* be the effective data type of the *j*-th column, for 1 (one) ≤ *j* ≤ *NCR*, as represented by the values of the TYPE, LENGTH, OCTET_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 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier param-

eter. TYPE is one of the code values in Table 14, "Codes used for implementation data types in SQL/CLI".

d) Let TDT_j be the effective data type of the *j*-th <target specification>, for 1 (one) ≤ *j* ≤ *NCR*, as represented by the values of the TYPE, LENGTH, OCTET_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 be set by separate invocations of the `GetDescriptor()` 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. TYPE either indicates ROW or is one of the code values in Table 15, "Codes used for application data types in SQL/CLI".

e) For every DT_j and TDT_j, 1 (one) ≤ *j* ≤ *NCR*:

   i) If DT_j is an array type and TDT_j is not an array locator type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

   ii) If DT_j is a multiset type and TDT_j is not a multiset locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

   iii) If DT_j is a row type, then

      Case:

      1) If TDT_j is not a row type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

      2) If TDT_j is a row type and DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

   iv) If DT_j and TDT_j are predefined types, then let *HL* be the programming language in which the invoking SQL-server is written. Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 19.5, "SQL/CLI 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".

      Case:

      1) If the row that contains the SQL data type corresponding to DT_j in the SQL data type column of the operative data type correspondence table contains "None" in the host data type column, and TDT_j is not a character string type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

2) Otherwise, if DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

v) If DT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

f) Let *NCP* 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 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter.

g) Let PDT_j be the effective data type of the *j*-th parameter, for 1 (one) ≤ *j* ≤ *NCP*, as represented by the values of the TYPE, LENGTH, OCTET_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 be returned 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. TYPE is one of the code values in Table 14, "Codes used for implementation data types in SQL/CLI".

h) Let $PTDT_j$ be the effective data type of the *j*-th <target specification>, for 1 (one) ≤ *j* ≤ *NCP*, as represented by the values of the TYPE, LENGTH, OCTET_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 be returned by separate invocations of the `GetDescriptor()` 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_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 from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. TYPE either indicates ROW or is one of the code values in Table 15, "Codes used for application data types in SQL/CLI".

i) For every PDT_j and $PTDT_j$, 1 (one) ≤ *j* ≤ *NCP*:

i) If PDT_j is an array data type and $PTDT_j$ is not an array locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

ii) If PDT_j is a multiset data type and $PTDT_j$ is not a multiset locator data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

iii) If PDT_j is a row data type, then

Case:

1) If $PTDT_j$ is not a row data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

2) If $PTDT_j$ is a row data type and PDT_j and $PTDT_j$ do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

iv) If PDT_j and $PTDT_j$ are predefined data types, then let *HL* be the programming language in which the invoking SQL-server is written. Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 19.5, "SQL/CLI 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".

Case:

1) If the row that contains the SQL data type corresponding to PDT_j in the SQL data type column of the operative data type correspondence table contains "None" in the host data type column, and $PTDT_j$ is not a character string type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

2) Otherwise, if PDT_j and $PTDT_j$ do not conform to the Syntax Rules of Subclause 9.24, "Data type identity", in [ISO9075-2], then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

v) If PDT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors*.

6) Case:

a) If the foreign server request associated with *EH* returns a set of rows, then:

i) The General Rules of Subclause 21.5, "Implicit EXECUTE USING and OPEN USING clauses", are applied to 'OPEN' and *S* as *TYPE* and *ALLOCATED FDW-EXECUTION*, respectively.

ii) The General Rules of Subclause 21.2, "Implicit foreign-data wrapper cursor", are applied to *S* as *ALLOCATED FDW-EXECUTION*.

b) Otherwise, the General Rules of Subclause 21.5, "Implicit EXECUTE USING and OPEN USING clauses", are applied to 'EXECUTE' and *S*, as *TYPE* and *ALLOCATED FDW-EXECUTION*, respectively.

7) *EH* is said to be an *opened FDW-execution*.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains Open.

22.3.34 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: *FDW-specific condition — invalid handle*.

3) The General Rules of Subclause 22.3.33, "Open", are applied with *EH* as the ExecutionHandle parameter.

Conformance Rules

1) Without Feature M010, "Foreign-data wrapper support", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains ReOpen.

22.3.35 TransmitRequest

Function

Transmit a foreign server request to be analyzed by the foreign server.

Definition

```
TransmitRequest (
    FSConnectionHandle  IN      INTEGER,
    RequestString       IN      CHARACTER VARYING (L),
    StringLength        IN      INTEGER,
    ExecutionHandle     OUT     INTEGER )
RETURNS SMALLINT
```

where: *L* is equal to the value of StringLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

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: *FDW-specific condition — invalid handle*.

3) Let *C* be the allocated FS-connection identified by *FSCH*.

4) The foreign-data wrapper diagnostics area associated with *C* is emptied.

5) Let *FR* be the foreign server request associated with the RequestString.

6) If the implementation-defined 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: *FDW-specific condition — limit on number of handles exceeded*.

7) Case:

   a) If the memory requirements to manage an FDW-execution cannot be satisfied, then ReplyHandle is set to zero and an exception condition is raised: *FDW-specific condition — memory allocation error*.

   b) If the resources to manage an FDW-execution 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-execution are allocated and are referred to as an *allocated FDW-execution*. The allocated FDW-execution is assigned a unique value *RHV* that is returned in ExecutionHandle.

8) Case:

   a) If the foreign-data wrapper cannot create an FDW-execution that corresponds to *FR*, then an exception condition is raised: *FDW-specific condition — unable to create reply*.

   b) Otherwise, the FDW-execution corresponding to *FR* is created.

9) The PASSTHROUGH flag associated with the allocated FDW-execution is set to *True*.

10) Let *SRDItemDescriptorAreas* be the number of item descriptor areas that need to be set up for the server row descriptor. Let *SRDHandle* be the DescriptorHandle that is returned by invocation of the `AllocDescriptor()` with *SRDItemDescriptorAreas* as the MaxDetailAreas parameter. Let *SRD* be the server row descriptor identified by *SRDHandle*. *SRD* is associated with the allocated FDW-execution.

    For this descriptor, fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", are set to the specified default values by invocation of the `SetDescriptor()` with *SRDHandle* as the DescriptorHandle parameter and *r* as the RecordNumber parameter, 1 (one) ≤ *r* ≤ *SRDItemDescriptorAreas*, and the code for the fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. All other fields in the item descriptor areas of *SRD* are initially undefined.

11) Let *SPDItemDescriptorAreas* be the number of item descriptor areas that need to be set up for the server parameter descriptor. Let *SPDHandle* be the DescriptorHandle that is returned by invocation of the `AllocDescriptor()` with *SPDItemDescriptorAreas* as the MaxDetailAreas parameter. Let *SPD* be the server parameter descriptor identified by *SPDHandle*. *SPD* is associated with the allocated FDW-execution.

    For this descriptor, fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", are set to the specified default values by invocation of the `SetDescriptor()` with *SPDHandle* as the DescriptorHandle parameter and *r* as the RecordNumber parameter, 1 (one) ≤ *r* ≤ *SPDItemDescriptorAreas*, and the code for the fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. All other fields in the item descriptor areas of *SPD* are initially undefined.

12) Let *WRDItemDescriptorAreas* be the number of item descriptor areas that need to be set up for the wrapper row descriptor. Let *WRDHandle* be the DescriptorHandle that is returned by invocation of the `AllocDescriptor()` with *WRDItemDescriptorAreas* as the MaxDetailAreas parameter. Let *WRD* be the wrapper row descriptor identified by *WRDHandle*. *WRD* is associated with the allocated FDW-execution.

    For this descriptor, fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", are set to the specified default values by invocation of the `SetDescriptor()` with *WRDHandle* as the DescriptorHandle parameter and *r* as the RecordNumber parameter, 1 (one) ≤ *r* ≤ *WRDItemDescriptorAreas*, and the code for the fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. All other fields in the item descriptor areas of *WRD* are initially undefined.

13) Let *WPDItemDescriptorAreas* be the number of item descriptor areas that need to be set up for the wrapper parameter descriptor. Let *WPDHandle* be the DescriptorHandle that is returned by invocation of the `AllocDescriptor()` with *WPDItemDescriptorAreas* as the MaxDetailAreas parameter. Let *WPD* be the wrapper parameter descriptor identified by *WPDHandle*. *WPD* is associated with the allocated FDW-execution.

    For this descriptor, fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", are set to the specified default values by invocation of the `SetDescriptor()` with *WPDHandle* as the DescriptorHandle parameter and *r* as the RecordNumber parameter, 1 (one) ≤ *r* ≤ *WPDItemDescriptorAreas*, and the code for the fields with non-blank entries in Table 34, "Foreign-data wrapper descriptor field default values", from Table 30, "Codes used for foreign-data wrapper descriptor fields", as the FieldIdentifier parameter. All other fields in the item descriptor areas of *WPD* are initially undefined.

14) The General Rules of Subclause 21.4, "Implicit DESCRIBE OUTPUT USING clause", are applied with RequestString and *WRD* as *SOURCE* and *DESCRIPTOR*, respectively.

15) The General Rules of Subclause 21.3, "Implicit DESCRIBE INPUT USING clause", are applied with RequestString and *WPD* as *SOURCE* and *DESCRIPTOR*, respectively.

Conformance Rules

1) Without Feature M007, "TransmitRequest", a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains TransmitRequest.

22.4 Foreign-data wrapper interface SQL-server routines

22.4.1 AllocDescriptor

Function

Allocate a foreign-data wrapper descriptor area and assign a handle to it.

Definition

```
AllocDescriptor (
    MaxDetailAreas        IN       SMALLINT,
    DescriptorHandle      OUT      INTEGER )
RETURNS SMALLINT
```

General Rules

1) Let *MDA* be the value of MaxDetailAreas.

2) If the implementation-defined maximum number of foreign-data wrapper descriptor areas that can be allocated at one time has already been reached, then DescriptorHandle is set to 0 (zero) and an exception condition is raised: *FDW-specific condition — limit on number of handles exceeded*.

3) Case:

   a) If the memory requirements to manage a foreign-data wrapper descriptor area having *MDA* item descriptor areas cannot be satisfied, then DescriptorHandle is set to 0 (zero) and an exception condition is raised: *FDW-specific condition — memory allocation error*.

   b) If the resources to manage a foreign-data wrapper descriptor area cannot be allocated for implementation-defined reasons, then DescriptorHandle is set to 0 (zero) and an implementation-defined exception condition is raised.

   c) Otherwise, the resources to manage a foreign-data wrapper descriptor area are allocated and are referred to as an *allocated foreign-data wrapper descriptor area*. The allocated foreign-data wrapper descriptor area is assigned a unique value that is returned in DescriptorHandle.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains AllocDescriptor.

22.4.2 FreeDescriptor

Function

Release resources associated with a foreign-data wrapper descriptor area.

Definition

```
FreeDescriptor (
    DescriptorHandle   IN      INTEGER )
RETURNS SMALLINT
```

General Rules

1) Let *DH* be the value of DescriptorHandle.

2) If *DH* does not identify an allocated foreign-data wrapper descriptor area, then an exception condition is raised: *FDW-specific condition — invalid handle*.

3) Let *D* be the allocated foreign-data wrapper descriptor area identified by *DH*.

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

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains FreeDescriptor.

22.4.3 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* has a maximum value equal to the implementation-defined maximum 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: *FDW-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 21.7, "Character string retrieval", are applied to AuthorizationId, *AID*, *BL*, StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetAuthorizationId.

22.4.4 GetBoolVE

Function

Get a handle for a <boolean value expression> from the <where clause> of a query.

Definition

```
GetBoolVE (
    RequestHandle         IN      INTEGER,
    BoolVENumber          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: *FDW-specific condition — invalid handle*.

3) Let *BVEN* be the value of BoolVENumber.

4) If *BVEN* is less than 1 (one), then an exception condition is raised: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <boolean value expression>s simply contained in the <where clause> of *Q*.

7) If *BVEN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) ValueExpressionHandle is set to the value expression handle associated with the *BVEN*-th <boolean value expression>.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetBoolVE.

22.4.5 GetDescriptor

Function

Get the value of a field from a foreign-data wrapper 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 foreign-data wrapper 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 30, "Codes used for foreign-data wrapper descriptor fields", then an exception condition is raised: *FDW-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 30, "Codes used for foreign-data wrapper descriptor fields", that contains *FI*.

6) 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*.

7) 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: *FDW-specific condition — invalid descriptor field identifier*.

8) Let *IDA* be the foreign-data wrapper item descriptor area of *D* specified by *RN*.

9) If *TYPE* is 'HEADER', then header information from the descriptor area *D* is retrieved as follows.

   NOTE 79 — In the row of Table 32, "Ability to retrieve foreign-data wrapper descriptor fields", that contains *FI*, let *MBR* be the value in the column that contains the descriptor type of *D*. If *MBR* is 'No', then the effect on the retrieved value is implementation-dependent.

   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 30, "Codes used for foreign-data wrapper descriptor fields", then the value retrieved is the value of the descriptor header field identified by *FI*.

10) If *TYPE* is 'ITEM', then item information from the descriptor area *D* is retrieved as follows.

NOTE 80 — Let *MBR* be the value of the May Be Retrieved column in the row of Table 32, "Ability to retrieve foreign-data wrapper descriptor fields", that contains *FI* and the column that contains the descriptor type *D*. If *MBR* is 'No', then the effect on the retrieved value is implementation-dependent.

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 30, "Codes used for foreign-data wrapper descriptor fields", then the value retrieved is the value of the descriptor item field identified by *FI*.

11) Let *V* be the value retrieved.

12) If *FI* indicates a descriptor field whose row in Table 4, "Fields in foreign-data wrapper 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.

13) Let *BL* be the value of BufferLength.

14) If *FI* indicates a descriptor field whose row in Table 4, "Fields in foreign-data wrapper descriptor areas", contains a Data Type that is CHARACTER VARYING, then the General Rules of Subclause 21.7, "Character string retrieval", are applied with Value, *V*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetDescriptor.

22.4.6 GetDistinct

Function

Determine whether the supplied query specifies DISTINCT.

Definition

```
GetDistinct (
    RequestHandle          IN      INTEGER,
    IsDistinct             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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) Case:

   a) If the <select list> of *Q* specifies a <set quantifier> that specifies DISTINCT, then IsDistinct is set to 1 (one).

   b) Otherwise, IsDistinct is set to 0 (zero).

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetDistinct.

22.4.7 GetNumBoolVE

Function

Get the number of <boolean value expression>s simply contained in the <where clause> of a query.

Definition

```
GetNumBoolVE (
    RequestHandle          IN      INTEGER,
    NumberOfBoolVEs        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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) NumberOfBoolVEs is set to the number of <boolean value expression> elements simply contained in the <where clause> of *Q*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumBoolVE.

22.4.8 GetNumChildren

Function

Get the number of <value expression>s simply contained in the containing <value expression>.

Definition

```
GetNumChildren (
    ValueExpressionHandle IN      INTEGER,
    NumberOfChildren      OUT     SMALLINT )
RETURNS SMALLINT
```

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: *FDW-specific condition — invalid handle*.

3) NumberOfChildren is set to the number of <value expression>s simply contained in the <value expression> to which *VEH* refers.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumChildren.

22.4.9 GetNumOrderByElems

Function

Get the number of <value expression>s that are used to order the rows of the result of the query identified by the request handle provided.

Definition

```
GetNumOrderByElems (
    RequestHandle         IN      INTEGER,
    NumberOfOrderByElems  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: *FDW-specific condition — invalid handle*.

3) NumberOfOrderByElems is set to the number of <value expression>s required to order the rows of the result of the query associated with *RH*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumOrderByElems.

22.4.10 GetNumRoutMapOpts

Function

Get the number of generic options of a routine mapping.

Definition

```
GetNumRoutMapOpts (
    RoutineMappingHandle  IN      INTEGER,
    OptionCount           OUT     INTEGER )
RETURNS SMALLINT
```

General Rules

1) Let *RH* be the value of RoutineMappingHandle.

2) If *RH* does not identify an allocated routine mapping description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

3) OptionCount is set to the number of generic options of the routine mapping described by *RH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumRoutMapOpts.

22.4.11 GetNumSelectElems

Function

Get the number of <value expression>s in the <select list> of a query.

Definition

```
GetNumSelectElems (
    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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) NumberOfSelectListElements is set to the number of <value expression> elements simply contained in the <select list> of *Q*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumSelectElems.

22.4.12 GetNumServerOpts

Function

Get the number of generic options associated with the foreign server.

Definition

```
GetNumServerOpts (
    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: *FDW-specific condition — invalid handle*.

3) OptionCount is set to the number of generic options associated with *SH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumServerOpts.

22.4.13 GetNumTableColOpts

Function

Get the number of generic options of a column of a foreign table.

Definition

```
GetNumTableColOpts (
    TableReferenceHandle            IN      INTEGER,
    ColumnName                      IN      CHARACTER(L),
    NameLength                      IN      SMALLINT,
    OptionCount                     OUT     INTEGER )
RETURNS SMALLINT
```

where *L* 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: *FDW-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) Otherwise, an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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: *FDW-specific condition — column name not found*.

   b) Otherwise, OptionCount is set to the number of generic options of the column of the foreign table referenced by *TRH* whose name is *TCN*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumTableColOpts.

22.4.14 GetNumTableOpts

Function

Get the number of generic options of a foreign table.

Definition

```
GetNumTableOpts (
    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: *FDW-specific condition — invalid handle*.

3) OptionCount is set to the number of generic options of the foreign table referenced by *TRH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumTableOpts.

22.4.15 GetNumTableRefElems

Function

Get the number of <table reference>s contained in the <from clause> of a query.

Definition

```
GetNumTableRefElems (
    RequestHandle              IN      INTEGER,
    NumberOfTableReferences    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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) NumberOfTableReferenceElements is set to the number of <table reference> elements simply contained in the <from clause> of *Q*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumTableRefElems.

22.4.16 GetNumUserOpts

Function

Get the number of generic options of a user mapping.

Definition

```
GetNumUserOpts (
    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: *FDW-specific condition — invalid handle*.

3) OptionCount is set to the number of generic options of the user mapping described by *UH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumUserOpts.

22.4.17 GetNumWrapperOpts

Function

Get the number of generic options of a foreign-data wrapper.

Definition

```
GetNumWrapperOpts (
    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: *FDW-specific condition — invalid handle*.

3) OptionCount is set to the number of generic options of the foreign-data wrapper described by *WH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetNumWrapperOpts.

22.4.18 GetOrderByElem

Function

Get the handle for a <value expression> used to order the result of a query.

Definition

```
GetOrderByElem (
    RequestHandle         IN      INTEGER,
    OrderByNumber         IN      SMALLINT,
    ValueExpressionHandle OUT     INTEGER,
    OrderingSpec          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: *FDW-specific condition — invalid handle*.

3) Let *ON* be the value of OrderByNumber.

4) If *ON* is less than 1 (one), then an exception condition is raised: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <value expression>s required to order the result of *Q*.

7) If *ON* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) Let *VEH* be the value expression handle associated with the *ON*-th <value expression> associated with *Q*.

9) ValueExpressionHandle is set to *VEH*.

10) Let *OS* be the <ordering specification> associated with *VEH*.

    Case:

    a) If *OS* specifies ASC, then set OrderingSpec to –1 (one).

    b) Otherwise, set OrderingSpec to 1 (one).

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetOrderByElem.

22.4.19 GetRoutMapOpt

Function

Get the name and value of a specified generic option associated with a given routine mapping, given the option number.

Definition

```
GetRoutMapOpt (
    RoutineMappingHandle  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 each of *L1* and *L2* have a value equal to the implementation-defined maximum length of a variable-length character string.

General Rules

1) Let *RH* be the value of RoutineMappingHandle.

2) If *RH* does not identify an allocated routine mapping description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

3) Let *ON* be the value of OptionNumber.

4) Let *N* be the number of generic options associated with *RH*.

5) If *ON* is less than 1 (one), then an exception condition is raised: *FDW-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 *RH* is retrieved.

   a) Let *NAME* be the name of the generic option.

   b) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *OPTIONVALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetRoutMapOpt.

22.4.20 GetRoutMapOptName

Function

Get the value of a generic option associated with a given routine mapping, given the option name.

Definition

```
GetRoutMapOptName (
    RoutineMappingHandle  IN      INTEGER,
    OptionName            IN      CHARACTER(L1),
    BufferLength1         IN      SMALLINT,
    OptionValue           OUT     CHARACTER(L2),
    BufferLength2         IN      SMALLINT,
    StringLength2         OUT     SMALLINT )
RETURNS SMALLINT
```

where each of *L1* and *L2* have a value equal to the implementation-defined maximum length of a variable-length character string.

General Rules

1) Let *RH* be the value of RoutineMappingHandle.

2) If *RH* does not identify an allocated routine mapping description, then an exception condition is raised: *FDW-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) Otherwise, an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-specific condition — invalid option name*.

      ii) Otherwise, let *ON* be the first *N* characters of OptionName and let *TON* be the value of:

```
TRIM ( BOTH ' ' FROM 'ON' )
```

6) Case:

   a) If *TON* is equivalent to the name of a generic option associated with *RH*, then:

      i) Let *OPTIONVALUE* be the value of the generic option associated with *RH* whose name is equivalent to *TON*.

      ii) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetRoutMapOptName.

22.4.21 GetRoutineMapping

Function

Get the routine mapping handle for an allocated routine mapping description.

Definition

```
GetRoutineMapping (
    ValueExpressionHandle         IN      INTEGER,
    RoutineMappingHandle          OUT     INTEGER )
RETURNS SMALLINT
```

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: *FDW-specific condition — invalid handle*.

3) Let *RMH* be the handle for the allocated routine mapping description that is associated with *VEH*.

4) RoutineMappingHandle is set to *RMH*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetRoutineMapping.

22.4.22 GetSelectElem

Function

Get the handle of a <value expression> simply contained in the <select list> of a query.

Definition

```
GetSelectElem (
    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: *FDW-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: *FDW-specific condition — invalid option index*.

5) Let *Q* be the query associated with *RH*.

6) Let *N* be the number of <value expression> elements simply contained 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) ValueExpressionHandle is set to the value expression handle associated with the *SLEN*-th <value expression> associated with *Q*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetSelectElem.

22.4.23 GetSelectElemType

Function

Get the kind of a <value expression> in the <select list> of a query.

Definition

```
GetSelectElemType (
    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 allocated <value expression> description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

3) ValueExpressionType is set to the value of ValueExpressionKind that would be returned by invocation of `GetValueExpKind()` with *VEH* as the ValueExpressionHandle parameter.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetSelectElemType.

22.4.24 GetServerName

Function

Get the name of the foreign server associated with the provided server handle.

Definition

```
GetServerName (
    ServerHandle        IN      INTEGER,
    ServerName          OUT     CHARACTER(L),
    BufferLength        IN      SMALLINT,
    StringLength        OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* has a maximum value equal to (2*n*+1), where *n* is the implementation-defined length of an <identifier>.

NOTE 81 — The length (2*n*+1) supports the syntax of <foreign server name>, which is "<identifier><period><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: *FDW-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 21.7, "Character string retrieval", are applied to ServerName, *SN*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetServerName.

22.4.25 GetServerOpt

Function

Get the value of a generic option associated with a foreign server.

Definition

```
GetServerOpt (
    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 each of *L1* and *L2* 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: *FDW-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: *FDW-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 21.7, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *OPTIONVALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetServerOpt.

22.4.26 GetServerOptByName

Function

Get the value of a generic option associated with a foreign server.

Definition

```
GetServerOptByName (
    ServerHandle          IN      INTEGER,
    OptionName            IN      CHARACTER(L1),
    BufferLength1         IN      SMALLINT,
    OptionValue           OUT     CHARACTER(L2),
    BufferLength2         IN      SMALLINT,
    StringLength2         OUT     SMALLINT )
RETURNS SMALLINT
```

where each of *L1* and *L2* 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: *FDW-specific condition — invalid handle*.

3) Let *BL* be the value of Bufferlength1.

4) Case:

   a) If *BL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *L* be *BL*, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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 *SH*.

6) Case:

   a) If *TON* is equivalent to the name of a generic option associated with *SH*, then:

i) Let *OPTIONVALUE* be the value of the generic option associated with *SH* whose name is equivalent to *TON*.

ii) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetServerOptByName.

22.4.27 GetServerType

Function

Get the type of a foreign server.

Definition

```
GetServerType (
    ServerHandle        IN      INTEGER,
    ServerType          OUT     CHARACTER(L),
    BufferLength        IN      SMALLINT,
    StringLength        OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* 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: *FDW-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 21.7, "Character string retrieval", are applied to ServerType, *ST*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetServerType.

22.4.28 GetServerVersion

Function

Get the version of a foreign server.

Definition

```
GetServerVersion (
    ServerHandle          IN      INTEGER,
    ServerVersion         OUT     CHARACTER(L),
    BufferLength          IN      SMALLINT,
    StringLength          OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* 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: *FDW-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 21.7, "Character string retrieval", are applied to ServerVersion, *SV*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetServerVersion.

22.4.29 GetSQLString

Function

Get a character string representation of the query that is associated with a given request handle.

Definition

```
GetSQLString (
    RequestHandle          IN      INTEGER,
    StringFormat           IN      INTEGER,
    SQLString              OUT     CHARACTER VARYING(L),
    BufferLength           IN      INTEGER,
    StringLength           OUT     INTEGER )
RETURNS SMALLINT
```

where *L* is greater than or equal to the StringLength value returned, with a maximum value equal to the implementation-defined maximum length of a variable-length character string.

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: *FDW-specific condition — invalid handle*.

3) Let *Q* be the query associated with *RH*.

4) Let *SF* be the value of StringFormat.

5) If *SF* does not identify a value that is equal to any value in Table 35, "Codes used for the format of the character string transmitted by GetSQLString()", then an exception condition is raised: *FDW-specific condition — invalid string format*.

6) Case:

   a) If *SF* identifies an implementation-defined value, then let *QueryString* be the implementation-defined character string representation of *Q*.

   b) Otherwise, let *QueryString* be a character string conforming to the Format and Syntax Rules of Subclause 14.3, "<SQL procedure statement>".

7) The General Rules of Subclause 21.7, "Character string retrieval", are applied to SQLString, *QueryString*, BufferLength, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M006, "GetSQLString routine", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetSQLString.

22.4.30 GetTableColOpt

Function

Get the name and value of a generic option associated with a column of the foreign table reference by the supplied table reference handle, given an option number.

Definition

```
GetTableColOpt (
    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 each of *L*, *L1*, and *L2* 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: *FDW-specific condition — invalid handle*.

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *L* be *NL*, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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' )
      ```

5) 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: *FDW-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: *FDW-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) Let *NAME* be the name of the *ON*-th generic option associated with the column identified by *TCN*.

vi) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

vii) Let *VALUE* be the value of the *ON*-th generic option associated with the column identified by *TCN*.

viii) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableColOpt.

22.4.31 GetTableColOptByName

Function

Get the value of a generic option associated with a column of the foreign table referenced by the specified table reference handle, given the option name.

Definition

```
GetTableColOptByName (
    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 each of *L*, *L1*, and *L2* 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: *FDW-specific condition — invalid handle*.

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *L* be *NL*, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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' )
      ```

5) 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: *FDW-specific condition — column name not found*.

b) Otherwise:

i) Let *BL* be the value of BufferLength1.

ii) Case:

1) If *BL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

2) Otherwise, let *BL1* be *BL*, 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.

Case:

A) If *BNO* ≠ *BL1*, then an exception condition is raised: *FDW-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' )
```

iii) Case:

1) If *TON* is not equivalent to the name of a generic option associated with *TRH*, then an exception condition is raised: *FDW-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 21.7, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableColOptByName.

22.4.32 GetTableOpt

Function

Get the name and value of a generic option associated with the foreign table identified by the given table reference handle, given an option number.

Definition

```
GetTableOpt (
    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 each of *L1* and *L2* 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: *FDW-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: *FDW-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) Let *NAME* be the name of the *ON*-th generic option associated with *TRH*.

8) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

9) Let *VALUE* be the value of the *ON*-th generic option associated with *TRH*.

10) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableOpt.

22.4.33 GetTableOptByName

Function

Get the value of a generic option associated with the foreign table identified by the specified table reference handle, given the option name.

Definition

```
GetTableOptByName (
    TableReferenceHandle IN     INTEGER,
    OptionName           IN     CHARACTER(L1),
    BufferLength1        IN     SMALLINT,
    OptionValue          OUT    CHARACTER(L2),
    BufferLength2        IN     SMALLINT,
    StringLength2        OUT    SMALLINT )
RETURNS SMALLINT
```

where each of *L1* and *L2* 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: *FDW-specific condition — invalid handle*.

3) Let *NL* be the value of BufferLength1.

4) Case:

   a) If *NL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *L* be *NL*, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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) Case:

   a) If *TON* is not equivalent to the name of a generic option associated with *TRH*, then an exception condition is raised: *FDW-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 21.7, "Character string retrieval", are applied to OptionValue, *VALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableOptByName.

22.4.34 GetTableRefElem

Function

Get a <table reference> element from the <from clause> of a query, given a request handle and a table reference element number.

Definition

```
GetTableRefElem (
    RequestHandle                   IN      INTEGER,
    TableReferenceElementNumber     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: *FDW-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: *FDW-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) TableReferenceHandle is set to the table reference handle associated with the *TREN*-th <table reference> associated with *Q*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableRefElem.

22.4.35 GetTableRefElemType

Function

Get the type of a <table reference>, given its table reference handle.

Definition

```
GetTableRefElemType (
    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: *FDW-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

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableRefElemType.

22.4.36 GetTableRefTableName

Function

Get the table name of a TABLE_NAME <table reference> identified by the given table reference handle.

Definition

```
GetTableRefTableName (
    TableReferenceHandle      IN      INTEGER,
    TableName                 OUT     CHARACTER(L),
    BufferLength              IN      SMALLINT,
    StringLength              OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* 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: *FDW-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: *FDW-specific condition — invalid handle*.

4) Let *NAME* be the value of the table name of the <table reference> identified by *TRH*.

5) Let *BL* be the value of BufferLength.

6) The General Rules of Subclause 21.7, "Character string retrieval", are applied to TableName, *NAME*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableRefTableName.

22.4.37 GetTableServerName

Function

Get the name of the foreign server associated with the foreign table identified by the given table reference handle.

Definition

```
GetTableServerName (
    TableReferenceHandle IN     INTEGER,
    ServerName           OUT    CHARACTER(L),
    BufferLength         IN     SMALLINT,
    StringLength         OUT    SMALLINT )
RETURNS SMALLINT
```

where *L* 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: *FDW-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 21.7, "Character string retrieval", are applied to ServerName, *SN*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTableServerName.

22.4.38 GetTRDHandle

Function

Get the descriptor handle of the table reference descriptor associated with a given table reference handle.

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: *FDW-specific condition — invalid handle*.

3) TRDHandle is set to the descriptor handle of the table reference descriptor associated with *TRH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetTRDHandle.

22.4.39 GetUserOpt

Function

Get the name and value of a generic option associated with the user mapping identified by the specified user handle, given an option number.

Definition

```
GetUserOpt (
    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 each of *L1* and *L2* has 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: *FDW-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: *FDW-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 21.7, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *OPTIONVALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetUserOpt.

22.4.40 GetUserOptByName

Function

Get the value of a generic option associated with the user mapping associated with the specified user handle, given the option name.

Definition

```
GetUserOptByName (
    UserHandle                  IN      INTEGER,
    OptionName                  IN      CHARACTER(L1),
    BufferLength1               IN      SMALLINT,
    OptionValue                 OUT     CHARACTER(L2),
    BufferLength2               IN      SMALLINT,
    StringLength2               OUT     SMALLINT )
RETURNS SMALLINT
```

where each of *L1* and *L2* has 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: *FDW-specific condition — invalid handle*.

3) Let *NL* be the value of BufferLength1.

4) Case:

   a) If *NL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *L* be *NL*, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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) Case:

   a) If *TON* is equivalent to the name of a generic option associated with *UH*, then:

      i) Let *OPTIONVALUE* be the value of the generic option associated with *UH* whose name is equivalent to *TON*.

ii) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetUserOptByName.

22.4.41 GetValExprColName

Function

Get the column name of a COLUMN_NAME <value expression>, given its value expression handle.

Definition

```
GetValExprColName (
    ValueExpressionHandle       IN      INTEGER,
    ColumnName                  OUT     CHARACTER(L),
    BufferLength                IN      SMALLINT,
    StringLength                OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* has a maximum value 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: *FDW-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: *FDW-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 21.7, "Character string retrieval", are applied to ColumnName, *NAME*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetValExprColName.

22.4.42 GetValueExpDesc

Function

Get the handle for a value expression descriptor describing a <value expression>, given its value expression handle.

Definition

```
GetValueExpDesc (
    ValueExpressionHandle    IN      INTEGER,
    ValueExpDescriptorHandle OUT     INTEGER )
RETURNS SMALLINT
```

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: *FDW-specific condition — invalid handle*.

3) Let *VEDH* be the value expression descriptor handle associated with the value expression descriptor that describes the <value expression> identified by *VEH*.

4) ValueExpDescriptorHandle is set to *VEDH*.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetValueExpDesc.

22.4.43 GetValueExpKind

Function

Get the kind of a <value expression>, given its value expression handle.

Definition

```
GetValueExpKind (
    ValueExpressionHandle IN      INTEGER,
    ValueExpressionKind   OUT     SMALLINT )
RETURNS SMALLINT
```

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: *FDW-specific condition — invalid handle*.

3) Let *VEK* be the kind of the <value expression> associated with *VEH*.

   NOTE 82 — The permissible values of the kind of a <value expression> are listed in Table 28, "Codes used for <value expression> kinds".

4) ValueExpressionKind is set to *VEK*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetValueExpKind.

22.4.44 GetValueExpName

Function

Get the name associated with a <value expression>.

Definition

```
GetValueExpName (
    ReplyHandle               IN      INTEGER,
    ValueExpressionHandle     IN      INTEGER,
    BufferLength              IN      INTEGER,
    ValueExpressionNameLength OUT INTEGER,
    ValueExpressionName       OUT CHARACTER(L) )
RETURNS SMALLINT
```

where *L* has a maximum value 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: *FDW-specific condition — invalid handle*.

3) Let *VEN* be the name associated with *VEH*.

4) The General Rules of Subclause 21.7, "Character string retrieval", are applied to ValueExpressionName, *VEN*, BufferLength, and ValueExpressionNameLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetValueExpName.

22.4.45 GetValueExpTable

Function

Get the table reference handle with which the table associated with the <value expression> identified by the specified value expression handle is associated.

Definition

```
GetValueExpTable (
    ValueExpressionHandle IN      INTEGER,
    TableReferenceHandle  OUT INTEGER )
RETURNS SMALLINT
```

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: *FDW-specific condition — invalid handle*.

3) Let *TBL* be the table associated with the allocated <value expression> identified by *VEH*. TableReference-Handle is set to the table reference handle that identifies the table reference descriptor that describes *T*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetValueExpTable.

22.4.46 GetVEChild

Function

Get a handle for the <value expression>, identified by an ordinal position, simply contained in the <value expression> identified by the specified value expression handle.

Definition

```
GetVEChild (
    ValueExpressionHandle      IN      INTEGER,
    Index                      IN SMALLINT,
    ChildValueExpressionHandle OUT INTEGER )
RETURNS SMALLINT
```

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: *FDW-specific condition — invalid handle*.

3) Let *NOC* be the number of <value expression>s immediately contained in the <value expression> to which *VEH* refers.

4) Let *I* be the value of Index.

5) If *I* is less than 1 (one), then an exception condition is raised: *FDW-specific condition — invalid option index*.

6) If *I* is greater than *NOC*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

7) ChildValueExpressionHandle is set to the value expression handle associated with the *I*-th simply contained <value expression>.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetVEChild.

22.4.47 GetWrapperLibraryName

Function

Get the library name associated with the foreign-data wrapper identified by the specified wrapper handle.

Definition

```
GetWrapperLibraryName (
    WrapperHandle           IN      INTEGER,
    WrapperLibraryName      OUT     CHARACTER(L),
    BufferLength            IN      SMALLINT,
    StringLength            OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* 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: *FDW-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 21.7, "Character string retrieval", are applied to WrapperLibraryName, *WL*, BufferLength, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetWrapperLibraryName.

22.4.48 GetWrapperName

Function

Get the name of the foreign-data wrapper identified by a specified wrapper handle.

Definition

```
GetWrapperName (
    WrapperHandle           IN      INTEGER,
    WrapperName             OUT     CHARACTER(L),
    BufferLength            IN      SMALLINT,
    StringLength            OUT     SMALLINT )
RETURNS SMALLINT
```

where *L* has a maximum value equal to (2*n*+1), where *n* is the implementation-defined length of an <identifier>.

NOTE 83 — The length (2*n*+1) supports the syntax of <foreign server name>, which is "<identifier><period><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: *FDW-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 21.7, "Character string retrieval", are applied to WrapperName, *WN*, BufferLength, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetWrapperName.

22.4.49 GetWrapperOpt

Function

Get the name and value of a generic option associated with the foreign-data wrapper identified by the specified wrapper handle, given an option number.

Definition

```
GetWrapperOpt (
    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 each of *L1* and *L2* has 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: *FDW-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: *FDW-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 21.7, "Character string retrieval", are applied to OptionName, *NAME*, BufferLength1, and StringLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

   c) Let *OPTIONVALUE* be the value of the generic option.

   d) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

Conformance Rules

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetWrapperOpt.

22.4.50 GetWrapperOptByName

Function

Get the value of a generic option associated with the foreign-data wrapper identified by the specified wrapper handle, given the option name.

Definition

```
GetWrapperOptByName (
    WrapperHandle           IN      INTEGER,
    OptionName              IN      CHARACTER(L1),
    BufferLength1           IN      SMALLINT,
    OptionValue             OUT     CHARACTER(L2),
    BufferLength2           IN      SMALLINT,
    StringLength2           OUT     SMALLINT )
RETURNS SMALLINT
```

where each of *L1* and *L2* has 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: *FDW-specific condition — invalid handle*.

3) Let *NL* be the value of BufferLength1.

4) Case:

   a) If *NL* is not positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *L* be *NL*, 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.

      Case:

      i) If *NO* ≠ *L*, then an exception condition is raised: *FDW-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) Case:

   a) If *TON* is equivalent to the name of a generic option associated with *WH*, then:

      i) Let *OPTIONVALUE* be the value of the generic option associated with *WH* whose name is equivalent to *TON*.

ii) The General Rules of Subclause 21.7, "Character string retrieval", are applied to OptionValue, *OPTIONVALUE*, BufferLength2, and StringLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found*.

Conformance Rules

1) Without Feature M030, "SQL-server foreign data support", a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetWrapperOptByName.

22.4.51 SetDescriptor

Function

Set a field in the foreign-data wrapper descriptor area identified by the specified descriptor handle.

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 foreign-data wrapper descriptor area identified by DescriptorHandle and let *N* be the value of the COUNT field of *D*.

2) Let *HL1* be the host language in which the SQL-server is written and let *HL2* be the host language in which the foreign-data wrapper is written.

3) Let *FI* be the value of FieldIdentifier.

4) If *FI* is not one of the code values in Table 30, "Codes used for foreign-data wrapper descriptor fields", then an exception condition is raised: *FDW-specific condition — invalid descriptor field identifier*.

5) Let *RN* be the value of RecordNumber.

6) Let *TYPE* be the value of the Type column in the row of Table 30, "Codes used for foreign-data wrapper descriptor fields", that contains *FI*.

7) If TYPE is 'ITEM' and *RN* is less than 1 (one), then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

8) Let *IDA* be the item descriptor area of *D* specified by *RN*.

9) 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.

10) Information is set in *D*.

    NOTE 84 — Let *MBS* be the value of the May Be Set column in the row of Table 33, "Ability to set foreign-data wrapper descriptor fields", that contains *FI* in the column that contains the descriptor type *DT*. If *MBS* is 'No', then the effect on the field is implementation-dependent.

    Case:

    a) If *FI* indicates COUNT, then

       Case:

i) If the memory requirements to manage the foreign-data wrapper descriptor area cannot be satisfied, then an exception condition is raised: *FDW-specific condition — memory allocation error*.

ii) Otherwise, the count of the number of foreign-data wrapper item descriptor areas is set to the value of Value.

b) If *FI* indicates OCTET_LENGTH, then the value of the OCTET_LENGTH field of *IDA* is set to the value 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 DATA, then the value of the DATA field of *IDA* is set to the value of Value.

e) If *FI* indicates INDICATOR, then the value of the INDICATOR field of *IDA* is set to the value of Value.

f) If *FI* indicates RETURNED_CARDINALITY, then the value of the RETURNED_CARDINALITY field of *IDA* is set to the value of Value.

g) 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 positive, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length*.

2) Otherwise, let *L* be *BL*, let *FV* be the first *L* octets of Value, and let *TFV* be the value of

```
TRIM ( BOTH ' ' FROM 'FV' )
```

iii) 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 [ISO9075-2], and let *TFVL* be the length in characters of *TFV*.

iv) 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*.

v) 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*.

vi) The value of the field of *IDA* identified by *FI* is set to the value of *FV*.

h) Otherwise, the value of the field of *IDA* identified by *FI* is set to the value of Value.

11) 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 foreign-data wrapper 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, ARRAY LOCATOR, MULTISET, or MULTISET 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 foreign-data wrapper 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.

12) If TYPE is 'ITEM' and *RN* is greater than *N*, then the COUNT field of *D* is set to *RN*.

13) Case:

a) If *HL1* and *HL2* are both pointer-supporting languages, and if *FI* indicates TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, 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).

b) Otherwise, 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 value of the DATA field of *IDA* is set to 0 (zero).

14) 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 foreign-data wrapper item descriptor areas", then an exception condition is raised: *FDW-specific condition — inconsistent descriptor information*.

15) Let *V* be the value of Value.

16) 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 BINARY, BINARY VARYING, or BINARY LARGE OBJECT, then the LENGTH field of *IDA* is set to the maximum possible length in octets of the indicated data type.

        iii) If *V* indicates a <datetime type>, then the PRECISION field of *IDA* is set to 0 (zero).

        iv) If *V* indicates INTERVAL, then the DATETIME_INTERVAL_PRECISION field of *IDA* is set to 2.

        v) 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.

        vi) If *V* indicates SMALLINT, INTEGER, or BIGINT, 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, INTEGER, or BIGINT data types, respectively.

        vii) 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.

        viii) If *V* indicates DECFLOAT, then the PRECISION field of *IDA* is set to the implementation-defined default value for the precision of the DECFLOAT 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: *FDW-specific condition — invalid data type*.

17) 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.

18) 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

Case:

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

1) Without Feature M030, “SQL-server foreign data support”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains SetDescriptor.

22.5 Foreign-data wrapper interface general routines

22.5.1 GetDiagnostics

Function

Get information from a foreign-data wrapper 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 31, "Codes used for foreign-data wrapper handle types", then an exception condition is raised: *FDW-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: *FDW-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: *FDW-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: *FDW-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: *FDW-specific condition — invalid handle*.

   e) If *HT* indicates SERVER HANDLE and Handle does not identify an allocated foreign-server description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

   f) If *HT* indicates TABLEREFERENCE HANDLE and Handle does not identify an allocated table reference description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

   g) If *HT* indicates USER HANDLE and Handle does not identify an allocated user mapping description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

h) If *HT* indicates VALUEEXPRESSION HANDLE and Handle does not identify an allocated value expression description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

i) If *HT* indicates WRAPPER HANDLE and Handle does not identify an allocated foreign-data wrapper description, then an exception condition is raised: *FDW-specific condition — invalid handle*.

j) If *HT* indicates WRAPPERENV HANDLE and Handle does not identify an allocated FDW-environment, then an exception condition is raised: *FDW-specific condition — invalid handle*.

4) Let *DI* be the value of DiagIdentifier.

5) If *DI* is not one of the code values in Table 29, "Codes used for foreign-data wrapper diagnostic fields", then an exception condition is raised: *FDW-specific condition — invalid attribute value*.

6) Let *TYPE* be the value of the Type column in the row that contains *DI* in Table 29, "Codes used for foreign-data wrapper diagnostic fields"

7) Let *RN* be the value of RecordNumber.

8) Let *R* be the most recently executed foreign-data wrapper interface 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 85 — 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 4.17.4, "Return codes", specifies the code values and their meanings.

NOTE 86 — 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 foreign-data wrapper 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 87 — 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 code of the SQLSTATE value corresponding to the status condition. That value shall be 'ISO 9075' if the class code is fully defined in Subclause 26.1, "SQLSTATE", and shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined class code.

g) If *DI* indicates SUBCLASS_ORIGIN, then the value retrieved is the identification of the naming authority that defined the subclass code of the SQLSTATE value corresponding to the status condition. That value shall be 'ISO 9075' if the class code is fully defined in Subclause 26.1, "SQLSTATE", and shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined subclass code.

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 3, "Fields used in foreign-data wrapper 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: *FDW-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 *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) Otherwise, DiagInfo is set to the first *BL* octets of *V*.

Conformance Rules

1) Without Feature M031, “Foreign-data wrapper general routines”, a conforming foreign-data wrapper shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains AllocQueryContext.

2) Without Feature M031, “Foreign-data wrapper general routines”, a conforming SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> that contains a <foreign-data wrapper interface routine name> that contains GetDiagnostics.

23 Diagnostics management

This Clause modifies Clause 23, "Diagnostics management", in ISO/IEC 9075-2.

23.1 <get diagnostics statement>

This Subclause modifies Subclause 23.1, "<get diagnostics statement>", in ISO/IEC 9075-2.

Function

Get exception or completion condition information from the diagnostics area.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) *Table 36, "SQL-statement codes", modifies Table 37, "SQL-statement codes", in [ISO9075-2].*

Table 36 — SQL-statement codes

SQL-statement	Identifier	Code
All alternatives from ISO/IEC 9075-2		
<alter foreign-data wrapper statement>	ALTER FOREIGN DATA WRAPPER	120
<alter routine mapping statement>	ALTER ROUTINE MAPPING	130
<alter foreign server statement>	ALTER SERVER	108
<alter foreign table statement>	ALTER FOREIGN TABLE	104

SQL-statement	Identifier	Code
<alter user mapping statement>	ALTER USER MAPPING	123
<drop foreign-data wrapper statement>	DROP FOREIGN DATA WRAPPER	121
<drop foreign server statement>	DROP SERVER	110
<drop foreign table statement>	DROP FOREIGN TABLE	105
<drop routine mapping statement>	DROP ROUTINE MAPPING	131
<drop user mapping statement>	DROP USER MAPPING	124
<foreign-data wrapper definition>	CREATE FOREIGN DATA WRAPPER	119
<foreign server definition>	CREATE SERVER	107
<foreign table definition>	CREATE FOREIGN TABLE	103
<import foreign schema statement>	IMPORT FOREIGN SCHEMA	125
<routine mapping definition>	CREATE ROUTINE MAPPING	132
<set passthrough statement>	SET PASSTHROUGH	126
<user mapping definition>	CREATE USER MAPPING	122
Statements that are defined by a foreign-data wrapper	A character string value defined by a foreign-data wrapper different from the value associated with any other SQL-statement	*x*[1]

[1] An implementation-defined negative number different from the value associated with any other SQL-statement.

Conformance Rules

No additional Conformance Rules.

24 Information Schema

This Clause modifies Clause 5, "Information Schema", in ISO/IEC 9075-11.

24.1 ATTRIBUTES view

This Subclause modifies Subclause 5.11, "ATTRIBUTES view", in ISO/IEC 9075-11.

Function

Identify the attributes of user-defined types defined in this catalog that are accessible to a given user.

Definition

Add the following columns to the end of outermost select list of the view definition:

```
, D1.DATALINK_LINK_CONTROL, D1.DATALINK_INTEGRITY, D1.DATALINK_READ_PERMISSION,
D1.DATALINK_WRITE_PERMISSION, D1.DATALINK_RECOVERY, D1.DATALINK_UNLINK
```

Conformance Rules

1) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . ATTRIBUTES . DATALINK_INTEGRITY.

2) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . ATTRIBUTES . DATALINK_LINK_CONTROL.

3) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . ATTRIBUTES . DATALINK_READ_PERMISSION.

4) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . ATTRIBUTES . DATALINK_RECOVERY.

5) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . ATTRIBUTES . DATALINK_UNLINK.

6) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . ATTRIBUTES . DATALINK_WRITE_PERMISSION.

24.2 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 CO
    WHERE ( CO.TABLE_CATALOG, CO.TABLE_SCHEMA, CO.TABLE_NAME, CO.COLUMN_NAME )
       IN ( SELECT CP.TABLE_CATALOG, CP.TABLE_SCHEMA, CP.TABLE_NAME,
                   CP.COLUMN_NAME
              FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES AS CP
              WHERE ( CP.GRANTEE IN
                      ( 'PUBLIC', CURRENT_USER )
                    OR
                      CP.GRANTEE IN
                      ( SELECT ROLE_NAME
                        FROM ENABLED_ROLES ) ) )
                AND
                    CO.TABLE_CATALOG
                  = ( SELECT CATALOG_NAME
                      FROM INFORMATION_SCHEMA_CATALOG_NAME ) ;

GRANT SELECT ON TABLE COLUMN_OPTIONS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMN_OPTIONS.

24.3 COLUMNS view

This Subclause modifies Subclause 5.21, "COLUMNS view", in ISO/IEC 9075-11.

Function

Identify the columns of tables defined in this catalog that are accessible to a given user.

Definition

Add the following columns to the end of outermost select list of the view definition:

```
, DATALINK_LINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION,
DATALINK_WRITE_PERMISSION, DATALINK_RECOVERY, DATALINK_UNLINK
```

Conformance Rules

1) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMNS . DATALINK_INTEGRITY.

2) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMNS . DATALINK_LINK_CONTROL.

3) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMNS . DATALINK_READ_PERMISSION.

4) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMNS . DATALINK_RECOVERY.

5) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMNS . DATALINK_UNLINK.

6) Insert this CR Without Feature M001, "Datalinks", conforming SQL language shall not reference INFORMATION_SCHEMA . COLUMNS . DATALINK_WRITE_PERMISSION.

24.4 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
    WHERE ( FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME )
       IN ( SELECT FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME
            FROM INFORMATION_SCHEMA.FOREIGN_DATA_WRAPPERS ) ;

GRANT SELECT ON TABLE FOREIGN_DATA_WRAPPER_OPTIONS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference INFORMATION_SCHEMA . FOREIGN_DATA_WRAPPER_OPTIONS.

2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference INFORMATION_SCHEMA . FOREIGN_DATA_WRAPPER_OPTIONS.