
**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)

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2023

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword.....	xiii
Introduction.....	xv
1 Scope.....	1
2 Normative references.....	2
3 Terms and definitions.....	3
4 Concepts.....	6
4.1 Notations and conventions.....	6
4.1.1 Notations.....	6
4.2 Data types.....	6
4.2.1 Naming of predefined types.....	6
4.2.2 Data type terminology.....	6
4.3 Columns, fields, and attributes.....	6
4.4 Tables.....	7
4.4.1 Introduction to tables.....	7
4.4.2 Base tables.....	7
4.4.2.1 Foreign tables.....	7
4.4.3 Unique identification of tables.....	7
4.4.4 Table descriptors.....	7
4.4.5 Syntactic analysis of derived tables and cursors.....	7
4.5 Functional dependencies.....	8
4.5.1 Known functional dependencies in a foreign table.....	8
4.6 SQL-schemas.....	8
4.7 SQL-statements.....	8
4.7.1 SQL-statements classified by function.....	8
4.7.1.1 SQL-schema statements.....	8
4.7.1.2 SQL-session statements.....	9
4.8 Basic security model.....	9
4.8.1 Privileges.....	9
4.9 SQL-transactions.....	9
4.9.1 Properties of SQL-transactions.....	9
4.10 SQL-sessions.....	9
4.10.1 SQL-session properties.....	9
4.11 Introduction to SQL/CLI.....	10
4.12 Foreign servers.....	10
4.13 Foreign-data wrappers.....	11
4.14 User mappings.....	12
4.15 Routine mappings.....	12
4.16 Generic options.....	13

4.17	Capabilities and options information.	13
4.18	Datalinks.	14
4.18.1	Introduction to datalinks.	14
4.18.2	Operations involving datalinks.	17
4.18.2.1	Operators that operate on datalinks.	17
4.18.2.2	Other operators involving datalinks.	18
4.19	Foreign-data wrapper interface.	18
4.19.1	Introduction to foreign-data wrapper interface.	18
4.19.2	Handles.	18
4.19.3	Foreign server sessions.	20
4.19.4	Foreign-data wrapper interface routines.	20
4.19.4.1	Introduction to foreign-data wrapper interface routines.	20
4.19.4.2	Handle routines.	20
4.19.4.3	Initialization routines.	24
4.19.4.4	Access routines.	24
4.19.4.5	Termination routines.	25
4.19.4.6	Decomposition and pass-through modes.	26
4.19.4.7	Sequence of actions during the execution of foreign server requests.	26
4.19.5	Return codes.	38
4.19.6	Foreign-data wrapper diagnostics areas.	38
4.19.7	Null pointers.	40
4.19.8	Foreign-data wrapper descriptor areas.	40
5	Lexical elements.	44
5.1	<token> and <separator>.	44
5.2	Names and identifiers.	46
6	Scalar expressions.	48
6.1	<data type>.	48
6.2	<cast specification>.	51
6.3	<value expression>.	52
6.4	<string value function>.	53
6.5	<datalink value expression>.	57
6.6	<datalink value function>.	58
7	Query expressions.	61
7.1	<table reference>.	61
8	Additional common rules.	68
8.1	Retrieval assignment.	68
8.2	Store assignment.	69
8.3	Result of data type combinations.	70
8.4	Type precedence list determination.	71
8.5	Determination of identical values.	72
8.6	Equality operations.	73
8.7	Grouping operations.	74
8.8	Multiset element grouping operations.	75
8.9	Ordering operations.	76
9	Additional common elements.	77
9.1	<generic options>.	77

9.2	<alter generic options>.....	78
10	Schema definition and manipulation.....	80
10.1	<schema definition>.....	80
10.2	<drop schema statement>.....	81
10.3	<table definition>.....	82
10.4	<unique constraint definition>.....	83
10.5	<check constraint definition>.....	84
10.6	<alter column data type clause>.....	85
10.7	<drop column definition>.....	86
10.8	<domain definition>.....	87
10.9	<assertion definition>.....	88
10.10	<user-defined type definition>.....	89
10.11	<SQL-invoked routine>.....	90
10.12	<drop routine statement>.....	91
10.13	<user-defined cast definition>.....	92
10.14	<user-defined ordering definition>.....	93
10.15	<foreign table definition>.....	94
10.16	<alter foreign table statement>.....	97
10.17	<add basic column definition>.....	98
10.18	<alter basic column definition>.....	99
10.19	<drop basic column definition>.....	100
10.20	<drop foreign table statement>.....	102
11	Access control.....	104
11.1	<privileges>.....	104
11.2	<revoke statement>.....	105
11.3	<user mapping definition>.....	106
11.4	<alter user mapping statement>.....	108
11.5	<drop user mapping statement>.....	109
12	SQL-client modules.....	110
12.1	<SQL-client module definition>.....	110
12.2	<externally-invoked procedure>.....	111
12.3	<SQL procedure statement>.....	114
12.4	Data type correspondences.....	115
13	Additional data manipulation rules.....	118
13.1	Effect of deleting rows from base tables.....	118
13.2	Effect of inserting tables into base tables.....	119
13.3	Effect of replacing rows in base tables.....	121
14	Session management.....	123
14.1	<set passthrough statement>.....	123
15	Dynamic SQL.....	125
15.1	Description of SQL descriptor areas.....	125
15.2	<prepare statement>.....	126
15.3	<deallocate prepared statement>.....	128
15.4	<describe statement>.....	129
15.5	<input using clause>.....	131

15.6	<output using clause>.....	135
15.7	<execute statement>.....	138
15.8	<dynamic declare cursor>.....	139
15.9	<allocate extended dynamic cursor statement>.....	140
15.10	<allocate received cursor statement>.....	141
15.11	<dynamic open statement>.....	142
15.12	<dynamic fetch statement>.....	143
15.13	<dynamic close statement>.....	144
16	Embedded SQL.....	145
16.1	<embedded SQL Ada program>.....	145
16.2	<embedded SQL C program>.....	147
16.3	<embedded SQL COBOL program>.....	148
16.4	<embedded SQL Fortran program>.....	149
16.5	<embedded SQL MUMPS program>.....	150
16.6	<embedded SQL Pascal program>.....	151
16.7	<embedded SQL PL/I program>.....	152
17	Diagnostics management.....	153
17.1	<get diagnostics statement>.....	153
18	Call-Level Interface specifications.....	155
18.1	<CLI routine>.....	155
18.2	Implicit DESCRIBE USING clause.....	156
18.3	Description of CLI item descriptor areas.....	157
18.4	Other tables associated with CLI.....	158
18.5	SQL/CLI data type correspondences.....	160
19	SQL/CLI routines.....	162
19.1	BuildDataLink().....	162
19.2	GetDataLinkAttr().....	163
19.3	GetInfo().....	165
20	URLs.....	166
20.1	URL format.....	166
21	Catalog manipulation.....	169
21.1	<foreign server definition>.....	169
21.2	<alter foreign server statement>.....	171
21.3	<drop foreign server statement>.....	172
21.4	<foreign-data wrapper definition>.....	174
21.5	<alter foreign-data wrapper statement>.....	176
21.6	<drop foreign-data wrapper statement>.....	177
21.7	<import foreign schema statement>.....	178
21.8	<routine mapping definition>.....	180
21.9	<alter routine mapping statement>.....	181
21.10	<drop routine mapping statement>.....	182
22	SQL/MED common specifications.....	183
22.1	Description of foreign-data wrapper item descriptor areas.....	183
22.2	Implicit foreign-data wrapper cursor.....	187
22.3	Implicit DESCRIBE INPUT USING clause for external data.....	189

22.4	Implicit DESCRIBE OUTPUT USING clause for external data.	192
22.5	Implicit EXECUTE USING and OPEN USING clauses for external data.	196
22.6	Implicit FETCH USING clause for external data.	199
22.7	Character string retrieval for external data.	203
22.8	Binary string retrieval for external data.	204
22.9	Tables used with SQL/MED.	205
23	Foreign-data wrapper interface routines.	216
23.1	<foreign-data wrapper interface routine>.	216
23.2	<foreign-data wrapper interface routine> invocation.	221
23.3	AdvanceInitRequest().	223
23.4	AllocQueryContext().	224
23.5	AllocWrapperEnv().	225
23.6	Close().	227
23.7	ConnectServer().	228
23.8	FreeExecutionHandle().	230
23.9	FreeFSConnection().	232
23.10	FreeQueryContext().	233
23.11	FreeReplyHandle().	234
23.12	FreeWrapperEnv().	235
23.13	GetNextReply().	236
23.14	GetNumReplyBoolVE().	237
23.15	GetNumReplyOrderBy().	238
23.16	GetNumReplySelectElems().	239
23.17	GetNumReplyTableRefs().	240
23.18	GetOpts().	241
23.19	GetReplyBoolVE().	243
23.20	GetReplyCardinality().	244
23.21	GetReplyDistinct().	245
23.22	GetReplyExecCost().	246
23.23	GetReplyFirstCost().	247
23.24	GetReplyOrderElem().	248
23.25	GetReplyReExecCost().	249
23.26	GetReplySelectElem().	250
23.27	GetReplyTableRef().	251
23.28	GetSPDHandle().	252
23.29	GetSRDHandle().	253
23.30	GetStatistics().	254
23.31	GetWPDHandle().	256
23.32	GetWRDHandle().	257
23.33	InitRequest().	258
23.34	Iterate().	262
23.35	Open().	264
23.36	ReOpen().	265
23.37	Execution of Open().	266
23.38	TransmitRequest().	270
23.39	AllocDescriptor().	273
23.40	FreeDescriptor().	274

23.41	GetAuthorizationId()	275
23.42	GetBoolVE()	276
23.43	GetDescriptor()	277
23.44	GetDistinct()	279
23.45	GetNumBoolVE()	280
23.46	GetNumChildren()	281
23.47	GetNumOrderByElems()	282
23.48	GetNumRoutMapOpts()	283
23.49	GetNumSelectElems()	284
23.50	GetNumServerOpts()	285
23.51	GetNumTableColOpts()	286
23.52	GetNumTableOpts()	288
23.53	GetNumTableRefElems()	289
23.54	GetNumUserOpts()	290
23.55	GetNumWrapperOpts()	291
23.56	GetOrderByElem()	292
23.57	GetRoutMapOpt()	293
23.58	GetRoutMapOptName()	295
23.59	GetRoutineMapping()	297
23.60	GetSelectElem()	298
23.61	GetSelectElemType()	299
23.62	GetServerName()	300
23.63	GetServerOpt()	301
23.64	GetServerOptByName()	303
23.65	GetServerType()	305
23.66	GetServerVersion()	306
23.67	GetSQLString()	307
23.68	GetTableColOpt()	308
23.69	GetTableColOptByName()	310
23.70	GetTableOpt()	312
23.71	GetTableOptByName()	313
23.72	GetTableRefElem()	315
23.73	GetTableRefElemType()	316
23.74	GetTableRefTableName()	317
23.75	GetTableServerName()	318
23.76	GetTRDHandle()	319
23.77	GetUserOpt()	320
23.78	GetUserOptByName()	322
23.79	GetValExprColName()	324
23.80	GetValueExpDesc()	325
23.81	GetValueExpKind()	326
23.82	GetValueExpName()	327
23.83	GetValueExpTable()	328
23.84	GetVEChild()	329
23.85	GetWrapperLibraryName()	330
23.86	GetWrapperName()	331
23.87	GetWrapperOpt()	332

23.88	GetWrapperOptByName().	334
23.89	SetDescriptor().	336
23.90	GetDiagnostics().	341
24	Information Schema.	345
24.1	Information Schema digital artifact.	345
24.2	ATTRIBUTES view.	345
24.3	COLUMN_OPTIONS view.	346
24.4	COLUMNS view.	347
24.5	FOREIGN_DATA_WRAPPER_OPTIONS view.	348
24.6	FOREIGN_DATA_WRAPPERS view.	349
24.7	FOREIGN_SERVER_OPTIONS view.	350
24.8	FOREIGN_SERVERS view.	351
24.9	FOREIGN_TABLE_OPTIONS view.	352
24.10	FOREIGN_TABLES view.	353
24.11	ROUTINE_MAPPING_OPTIONS view.	354
24.12	ROUTINE_MAPPINGS view.	355
24.13	USER_MAPPING_OPTIONS view.	356
24.14	USER_MAPPINGS view.	357
24.15	Short name views.	358
25	Definition Schema.	362
25.1	Definition Schema digital artifact.	362
25.2	COLUMN_OPTIONS base table.	362
25.3	DATA_TYPE_DESCRIPTOR base table.	364
25.4	FOREIGN_DATA_WRAPPER_OPTIONS base table.	371
25.5	FOREIGN_DATA_WRAPPERS base table.	372
25.6	FOREIGN_SERVER_OPTIONS base table.	373
25.7	FOREIGN_SERVERS base table.	374
25.8	FOREIGN_TABLE_OPTIONS base table.	375
25.9	FOREIGN_TABLES base table.	376
25.10	ROUTINE_MAPPING_OPTIONS base table.	377
25.11	ROUTINE_MAPPINGS base table.	378
25.12	SQL_SIZING base table.	379
25.13	TABLES base table.	380
25.14	USAGE_PRIVILEGES base table.	381
25.15	USER_MAPPING_OPTIONS base table.	382
25.16	USER_MAPPINGS base table.	383
26	Status codes.	384
26.1	SQLSTATE.	384
27	Conformance.	387
27.1	Claims of conformance to SQL/MED.	387
27.2	Additional conformance requirements for SQL/MED.	387
Annex A	(informative) SQL conformance summary.	390
Annex B	(informative) Implementation-defined elements.	408
Annex C	(informative) Implementation-dependent elements.	422
Annex D	(informative) SQL optional feature taxonomy.	427

Annex E (informative) Deprecated features	429
Annex F (informative) Incompatibilities with ISO/IEC 9075:2016	430
Annex G (informative) Defect Reports not addressed in this edition of this document	431
Annex H (informative) Example header files	432
H.1 C header file sqlcli.h.	432
H.2 COBOL library item SQLCLI.	433
Annex I (informative) SQL/MED model	435
Bibliography	439
Index	440

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

Tables

Table	Page
1 Valid datalink file control options.	16
2 Sequence of actions during the execution of foreign server requests.	26
3 Fields used in foreign-data wrapper diagnostics areas.	39
4 Fields in foreign-data wrapper descriptor areas.	41
5 Data type correspondences for Ada.	115
6 Data type correspondences for C.	115
7 Data type correspondences for COBOL.	116
8 Data type correspondences for Fortran.	116
9 Data type correspondences for M.	116
10 Data type correspondences for Pascal.	116
11 Data type correspondences for PL/I.	117
12 Codes used for SQL data types in Dynamic SQL.	125
13 SQL-statement codes.	153
14 Abbreviated SQL/CLI generic names.	155
15 Codes used for implementation data types in SQL/CLI.	157
16 Codes used for application data types in SQL/CLI.	157
17 Codes used to identify SQL/CLI routines.	158
18 Codes and data types for implementation information.	158
19 Codes used for datalink attributes.	158
20 Data types of attributes.	158
21 SQL/CLI data type correspondences for Ada.	160
22 SQL/CLI data type correspondences for C.	160
23 SQL/CLI data type correspondences for COBOL.	160
24 SQL/CLI data type correspondences for Fortran.	161
25 SQL/CLI data type correspondences for M.	161
26 SQL/CLI data type correspondences for Pascal.	161
27 SQL/CLI data type correspondences for PL/I.	161
28 Codes used for <table reference> types.	205
29 Codes used for <value expression> kinds.	205
30 Codes used for foreign-data wrapper diagnostic fields.	205
31 Codes used for foreign-data wrapper descriptor fields.	206
32 Codes used for foreign-data wrapper handle types.	208
33 Ability to retrieve foreign-data wrapper descriptor fields.	208
34 Ability to set foreign-data wrapper descriptor fields.	210
35 Foreign-data wrapper descriptor field default values.	213
36 Codes used for the format of the character string transmitted by GetSQLString().	215
37 SQLSTATE class and subclass codes.	384
38 Implied feature relationships of SQL/MED.	389
D.1 Feature taxonomy for optional features.	427
I.1 Legend for SQL/MED interfaces.	436
I.2 Legend for SQL/MED information flow.	437

Figures

Figure	Page
1 SQL/MED interfaces.	435
2 SQL/MED information flow.	437

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

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.

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 or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC have not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and <https://patents.iec.ch>. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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

For an explanation of the voluntary nature of standards, 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 www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This fifth edition cancels and replaces the fourth edition (ISO/IEC 9075-9:2016), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 9075-9:2016/Cor.1:2019 and ISO/IEC 9075-9:2016/Cor.2:2022.

The main changes are as follows:

- improve the presentation and accuracy of the summaries of implementation-defined and implementation-dependent aspects of this document;
- introduction of several digital artifacts;
- alignment with updated ISO house style and other guidelines for creating standards.

This fifth edition of ISO/IEC 9075-9 is designed to be used in conjunction with the following editions of other parts of the ISO/IEC 9075 series, all published in 2023:

- ISO/IEC 9075-1, sixth edition;
- ISO/IEC 9075-2, sixth edition;

ISO/IEC 9075-9:2023(E)

- ISO/IEC 9075-3, sixth edition;
- ISO/IEC 9075-4, seventh edition;
- ISO/IEC 9075-10, fifth edition;
- ISO/IEC 9075-11, fifth edition;
- ISO/IEC 9075-13, fifth edition;
- ISO/IEC 9075-14, sixth edition;
- ISO/IEC 9075-15, second edition;
- ISO/IEC 9075-16, first edition.

A list of all parts in the ISO/IEC 9075 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

Introduction

The organization of this document is as follows:

- 1) **Clause 1, “Scope”**, specifies the scope of this document.
- 2) **Clause 2, “Normative references”**, identifies additional standards that, through reference in this document, constitute provisions of this document.
- 3) **Clause 3, “Terms and definitions”**, defines the terms and definitions used in this document.
- 4) **Clause 4, “Concepts”**, presents concepts related to this document.
- 5) **Clause 5, “Lexical elements”**, defines the lexical elements of the language specified in this document.
- 6) **Clause 6, “Scalar expressions”**, defines the elements of the language that produce scalar values.
- 7) **Clause 7, “Query expressions”**, defines the elements of the language that produce rows and tables of data.
- 8) **Clause 8, “Additional common rules”**, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.
- 9) **Clause 9, “Additional common elements”**, defines additional common elements used in the definition of foreign tables, foreign servers, and foreign-data wrappers.
- 10) **Clause 10, “Schema definition and manipulation”**, defines facilities related to foreign tables and datalink type support for creating and managing a schema.
- 11) **Clause 11, “Access control”**, defines facilities for controlling access to SQL-data.
- 12) **Clause 12, “SQL-client modules”**, defines SQL-client modules and externally-invoked procedures.
- 13) **Clause 13, “Additional data manipulation rules”**, defines additional rules for data manipulation.
- 14) **Clause 14, “Session management”**, defines the SQL-session management statements.
- 15) **Clause 15, “Dynamic SQL”**, defines the dynamic SQL statements.
- 16) **Clause 16, “Embedded SQL”**, defines the embedded SQL statements.
- 17) **Clause 17, “Diagnostics management”**, defines the diagnostics management facilities.
- 18) **Clause 18, “Call-Level Interface specifications”**, defines facilities for using SQL through a Call-Level Interface.
- 19) **Clause 19, “SQL/CLI routines”**, defines each of the routines that comprise the Call-Level Interface.
- 20) **Clause 20, “URLs”**, specifies the format of URLs used in this document.
- 21) **Clause 21, “Catalog manipulation”**, defines facilities for creating, altering, and dropping foreign servers and foreign-data wrappers.
- 22) **Clause 22, “SQL/MED common specifications”**, specifies common facilities used by SQL/MED.
- 23) **Clause 23, “Foreign-data wrapper interface routines”**, specifies the interaction between an SQL-server and a foreign-data wrapper.
- 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 document 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 document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect 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 document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-dependent.
- 31) [Annex D, “SQL optional feature taxonomy”](#), is an informative Annex. It identifies the optional features of the SQL language specified in this document by an identifier and a short descriptive name. This taxonomy is used to specify conformance.
- 32) [Annex E, “Deprecated features”](#), is an informative Annex. It lists features that the responsible Technical Committee intends not to include in a future edition of this document.
- 33) [Annex F, “Incompatibilities with ISO/IEC 9075:2016”](#), is an informative Annex. It lists incompatibilities with the previous edition of this document.
- 34) [Annex G, “Defect Reports not addressed in this edition of this document”](#), is an informative Annex. It describes the Defect Reports that were known at the time of publication of this document. Each of these problems is a problem carried forward from the previous edition of this document. No new problems have been created in the drafting of this edition of this document.
- 35) [Annex H, “Example 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 document, [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.

Information technology — Database language SQL —

Part 9:

Management of External Data (SQL/MED)**1 Scope**

This document defines extensions to Database Language SQL to support management of external data through the use of foreign-data wrappers and datalink types.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

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

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

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

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

W3C XML 1.0 *Extensible Markup Language (XML) Version 1.0*, W3C Recommendation. Edited by: Bray, Tim et al. 26 November 2008, revised 7 February 2013
Available at: <https://www.w3.org/TR/xml>

W3C XML 1.1 *Extensible Markup Language (XML) Version 1.1*, W3C Recommendation. Edited by: Bray, Tim et al. 16 August 2006, revised 29 September 2006
Available at: <https://www.w3.org/TR/xml11>

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9075-1, ISO/IEC 9075-2, and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.1

Valid XML document

Valid XML 1.0 document (3.2) or *Valid XML 1.1 document* (3.3)

3.2

Valid XML 1.0 document

well-formed XML document that meets certain further constraints, as defined in XML 1.0

3.3

Valid XML 1.1 document

well-formed XML document that meets certain further constraints, as defined in XML 1.1

3.4

XML document

XML 1.0 document (3.5) or *XML 1.1 document* (3.6)

3.5

XML 1.0 document

well-formed XML document, as defined in XML 1.0

3.6

XML 1.1 document

well-formed XML document, as defined in XML 1.1

3.7

DTD

XML 1.0 document type declaration (3.8) or *XML 1.1 document type declaration* (3.9)

3.8

XML 1.0 document type declaration

document type declaration, as defined in XML 1.0

3.9

XML 1.1 document type declaration

document type declaration, as defined in XML 1.1

3.10

access token

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

Note 1 to entry: An access token is either a *read token* or a *write token*.

3.11

datalink

value, of data type DATALINK, referencing some file that is not part of the SQL-environment

Note 1 to entry: The file is assumed to be managed by some external file manager.

3.12

datalinker

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

3.13

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

3.14

foreign-data wrapper

named collection of routines, invocable by the SQL-server, supporting the programming interface specified for such routines in this document

3.15

foreign server

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

3.16

foreign server request

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

3.17

foreign table

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

Note 1 to entry: 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.

3.18

integrity control option

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

3.19

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)

3.20

read permission option

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

3.21

recovery option

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

3.22

routine mapping

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

3.23

unlink option

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

3.24

user mapping

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

3.25

write permission option

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

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

4 Concepts

This Clause modifies *Clause 4, “Concepts”, in ISO/IEC 9075-2.*

4.1 Notations and conventions

This Subclause modifies *Subclause 4.1, “Notations and conventions”, in ISO/IEC 9075-2.*

4.1.1 Notations

This Subclause modifies *Subclause 4.1.1, “Notations”, in ISO/IEC 9075-2.*

The notations used in this document are defined in ISO/IEC 9075-1.

The syntax defined in this document is available from the ISO website as a “digital artifact”. See <https://standards.iso.org/iso-iec/9075/-9/ed-5/en/> to download digital artifacts for this document. To download the syntax defined in a plain-text format, select the file named `ISO_IEC_9075-9(E)_MED.bnf.txt`. To download the syntax defined in an XML format, select the file named `ISO_IEC_9075-9(E)_MED.bnf.xml`.

4.2 Data types

This Subclause modifies *Subclause 4.2, “Data types”, in ISO/IEC 9075-2.*

4.2.1 Naming of predefined types

This Subclause modifies *Subclause 4.2.2, “Naming of predefined types”, in ISO/IEC 9075-2.*

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

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

4.2.2 Data type terminology

This Subclause modifies *Subclause 4.2.4, “Data type terminology”, in ISO/IEC 9075-2.*

Insert into the 11th paragraph, after the last list item:

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

4.3 Columns, fields, and attributes

This Subclause modifies *Subclause 4.15, “Columns, fields, and attributes”, in ISO/IEC 9075-2.*

Insert after the 8th 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 1 — 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.

Insert after the 8th paragraph: *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 $V1$.

Insert after the 8th paragraph: $V2$ 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 $V3$.

4.4 Tables

This Subclause modifies Subclause 4.17, “Tables”, in ISO/IEC 9075-2.

4.4.1 Introduction to tables

This Subclause modifies Subclause 4.17.1, “Introduction to tables”, in ISO/IEC 9075-2.

Augment the 3rd paragraph by adding “foreign table” to the list of table types.

4.4.2 Base tables

This Subclause modifies Subclause 4.17.2, “Base tables”, in ISO/IEC 9075-2.

4.4.2.1 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 4.13, “Foreign-data wrappers”).

4.4.3 Unique identification of tables

This Subclause modifies Subclause 4.17.5, “Unique identification of tables”, in ISO/IEC 9075-2.

- **Insert into the 1st paragraph, after the 4th list item:** The <table name> of a foreign table uniquely identifies a multiset of rows.

4.4.4 Table descriptors

This Subclause modifies Subclause 4.17.7, “Table descriptors”, in ISO/IEC 9075-2.

Augment the 1st paragraph by adding “foreign table descriptor” to the list of possible table descriptors..

Insert after the 6th paragraph: A foreign table descriptor describes a foreign table. In addition to the components of every table descriptor, a foreign table descriptor includes the following components.

- 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 2 — This document currently restricts foreign tables such that they are neither insertable-into nor updatable. It is possible that this restriction will not appear in future editions.

4.4.5 Syntactic analysis of derived tables and cursors

This Subclause modifies Subclause 4.17.8, “Syntactic analysis of derived tables and cursors”, in ISO/IEC 9075-2.

Insert into the 7th paragraph, after the 1st list item:

- If *TORQN* identifies a foreign table, then *TORQN* has no generally underlying table specifications.

4.5 Functional dependencies

This Subclause modifies Subclause 4.26, “Functional dependencies”, in ISO/IEC 9075-2.

4.5.1 Known functional dependencies in a foreign table

There are no rules in this document to determine known functional dependencies in a foreign table. However, implementation-defined rules may determine known functional dependencies, if any, in a foreign table.

4.6 SQL-schemas

This Subclause modifies Subclause 4.28, “SQL-schemas”, in ISO/IEC 9075-2.

Insert into the 2nd paragraph, after the 2nd list item:

- A foreign table descriptor.

Augment the 5th paragraph by adding “foreign tables” to the list of objects that are identified by <table name>s.

4.7 SQL-statements

This Subclause modifies Subclause 4.41, “SQL-statements”, in ISO/IEC 9075-2.

4.7.1 SQL-statements classified by function

This Subclause modifies Subclause 4.41.2, “SQL-statements classified by function”, in ISO/IEC 9075-2.

4.7.1.1 SQL-schema statements

This Subclause modifies Subclause 4.41.2.1, “SQL-schema statements”, in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

- <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>

4.7.1.2 SQL-session statements

This Subclause modifies Subclause 4.41.2.7, “SQL-session statements”, in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

- <set passthrough statement>

4.8 Basic security model

This Subclause modifies Subclause 4.42, “Basic security model”, in ISO/IEC 9075-2.

4.8.1 Privileges

This Subclause modifies Subclause 4.42.2, “Privileges”, in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

- foreign table
- foreign-data wrapper
- foreign server

NOTE 3 — 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 can raise exceptions.

Insert into the 11th paragraph, after the last list item:

- foreign-data wrapper
- foreign server

4.9 SQL-transactions

This Subclause modifies Subclause 4.43, “SQL-transactions”, in ISO/IEC 9075-2.

4.9.1 Properties of SQL-transactions

This Subclause modifies Subclause 4.43.3, “Properties of SQL-transactions”, in ISO/IEC 9075-2.

Augment the 3rd paragraph by adding “foreign tables” to the list of tables.

4.10 SQL-sessions

This Subclause modifies Subclause 4.45, “SQL-sessions”, in ISO/IEC 9075-2.

4.10.1 SQL-session properties

This Subclause modifies Subclause 4.45.3, “SQL-session properties”, in ISO/IEC 9075-2.

Insert before the 13th 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 into the 13th paragraph, after the last list item:

- 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 before the 13th paragraph: At the end of every SQL-session, every FSConnection handle that is contained in the SQL-session context is freed.

Insert before the 13th paragraph: At the end of every SQL-session, every WrapperEnv handle that is contained in the SQL-session context is freed.

Insert before the 13th 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.

4.11 Introduction to SQL/CLI

This Subclause modifies Subclause 4.2, “Introduction to SQL/CLI”, in ISO/IEC 9075-3.

Insert after the 22nd 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.

4.12 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 4.13, “Foreign-data wrappers”).

NOTE 4 — The abbreviation “FDW” is sometimes used to evoke the notion of a “foreign-data wrapper”.

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 the following.

- 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 (IV126).

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 a subset of the <query specification> statement, 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 5 — Some SQL-aware foreign servers can 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 can concurrently respond to SQL-clients of their own; this does not change the relationships specified in this document.

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 ISO/IEC 9075-11.

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.

ISO/IEC 9075-2 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 (UW006).

4.13 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 the following.

- 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>.

4.14 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 the following.

- 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>.

4.15 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 following

- 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>.

4.16 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 document. 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.

4.17 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 document 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 DTD (XML Document Type Declaration) that is either internal to the XML document or external (requiring that it be available to the SQL-server in an implementation-defined (IW014) manner).

NOTE 6 — This edition of this document specifies the use of a DTD. It is possible that future editions will specify the use of an XML Schema, either as an alternative to a DTD or instead of a DTD.

4.18 Datalinks

4.18.1 Introduction to 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 the following.

- 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 (UV032) 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 document. The character set of the File Reference, referred to as the *datalink character set* is implementation-defined (IV127).

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 document 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 (UW007). 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 (UA072). 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 (IV013).

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 7 — 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 given here.

- NO LINK CONTROL: Although every File Reference shall conform to the Format and Syntax Rules of Subclause 20.1, “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.

4.18 Datalinks

- 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 8 — “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

Integrity	Read permission	Write permission	Recovery	Unlink
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 9 — 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 the following (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 (IL018) *maximum datalink length* determines the amount of space, in octets, that is allocated for the following objects.

- 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.

4.18.2 Operations involving datalinks

4.18.2.1 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 10 — “host”, “scheme”, and “path” are defined in Subclause 6.6, “<datalink value function>”.

4.18.2.2 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 (UW008) fashion.

4.19 Foreign-data wrapper interface

4.19.1 Introduction to 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.

4.19.2 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 (UA012) 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 11 — “decomposition mode” and “pass-through mode” are defined in Subclause 4.19.4.6, “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.

4.19.3 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)`.

4.19.4 Foreign-data wrapper interface routines

4.19.4.1 Introduction to 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 SQL-implementation are either all functions or all procedures, the choice being implementation-defined (IA099). 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.

4.19.4.2 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 a 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.

4.19.4.3 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 (UA033) 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.

4.19.4.4 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 (UA034) 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.

4.19.4.5 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

4.19 Foreign-data wrapper interface

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.

4.19.4.6 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.

4.19.4.7 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.		⇒		

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
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 12 — 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 13 — 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.				
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 14 — This information is provided in the <foreign server definition>.					
12	Executes the Get... (ServerHandle, ...) routines as requested from the foreign data wrapper.		⇒		

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
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 15 — 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 16 — If the current SQL-session context already includes a {foreign server name : FSConnectionHandle} pair that could be used, then Step 8 through Step 17 are optional.					
18	Invokes the Alloc-QueryContext() routine in the foreign-data wrapper to obtain a Query-ContextHandle .		⇒		
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.				

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
21	Creates as many TableReferenceHandles 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.				
23	Creates a Wrapper-Parameter-Descriptor (WPD) to describe the <dynamic parameter specification>s in the <query specification> described by the RequestHandle , and sets all the fields with details about each of the <dynamic parameter specification>s.				

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
24	Creates as many ValueExpressionHandles 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.	⇒		
26			⇐	Invokes the Get... (RequestHandle, ...), GetTRDHandle (TableReferenceHandle), 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.

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
NOTE 17 — 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 Get-Descriptor (TRD) routines.					
27	Executes the Get... (RequestHandle, ...) , GetTRDHandle (TableReferenceHandle) , Get-Descriptor (TRD) , Get... (TableReferenceHandle, ...) , and Get... (ValueExpressionHandle, ...) routines as requested by the foreign-data wrapper.		⇒		
NOTE 18 — 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 Get-Descriptor (TRD) routines.					
28				Creates a Reply-Handle 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.	

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
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.		⇒		
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 .

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
33		Executes the Alloc-Descriptor () routine as requested by the foreign-data wrapper.	⇒		
34				Repeats steps Step 25 through Step 33 to create multiple Reply-Handle / ExecutionHandle pairs for the same foreign server request.	
NOTE 19 — Step 34 is optional.					
35	Invokes the Get...(Reply-Handle, ...) 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.		⇒		
36			⇐	Executes the Get...(Reply-Handle, ...) 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 Get-NextReply() routine and returns a ReplyHandle and ExecutionHandle .	
39	Repeats Step 35 through Step 38 .				

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
NOTE 20 — Step 37 through Step 38 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 21 — Step 42 is optional.					
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.				

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
NOTE 22 — In pass-through mode, SetDescriptor (SRD) will only be invoked if results are returned by the foreign-data server.					
47	Invokes the GetSP-DHandle (ExecutionHandle) routine to get the SPD.	Invokes the GetWP-DHandle (ExecutionHandle) and GetSPDHandle (ExecutionHandle) routines to get the WPD and SPD, respectively.	⇒		
48			⇒	Executes the GetSP-DHandle (ExecutionHandle) routine as requested by the SQL-server.	Executes the Get-WPDHandle (ExecutionHandle) and GetSPDHandle (ExecutionHandle) routines as requested by the SQL-server.
49		Invokes Get-Descriptor (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 TableReferenceHandles .				
53	Frees each of the ValueExpressionHandles .				
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.	

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
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 23 — In pass-through mode, Step 56 through Step 58 are executed only if the foreign-data wrapper returns a set of rows.					
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.	

Step	SQL-server		Flow	Foreign-data wrapper	
	Decomposition	Pass-through		Decomposition	Pass-through
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.	
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 24 — 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 25 — Step 69 and Step 70 are optional, if the SQL-server wants to reuse the WrapperEnvHandle.					

4.19.5 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 26 — Foreign-data wrapper functions and foreign-data wrapper procedures are defined in Subclause 23.1, “<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 (01000)*.
- A value of 100 indicates **No data found**. The foreign-data wrapper interface routine executed successfully but a completion condition was raised: *no data (02000)*.
- A value of –1 (negative one) indicates **Error**. The foreign-data wrapper interface routine did not execute successfully. An exception condition other than *FDW-specific condition — invalid handle (HV00B)* 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 (HV00B)*.

If the foreign-data wrapper interface routine did not execute successfully, then the values of all output arguments are implementation-dependent (UV034) unless explicitly defined by this document.

In addition to providing the return code, for all foreign-data wrapper interface routines other than `GetDiagnostics()`, the SQL-implementation records information about completion conditions and about exception conditions raised other than *FDW-specific condition — invalid handle (HV00B)* 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.

4.19.6 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 (HV00B)*, 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 '02nnn', where 'nnn' is an implementation-defined (IC001) subclass code.

Status records in the diagnostics area are placed in an order that is implementation-dependent (US025) 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* (02000), which in turn have precedence over status records corresponding to the completion condition *warning* (01000);
- apart from any status records corresponding to an implementation-defined *no data* (02000), any status record corresponding to an implementation-defined condition that duplicates, in whole or in part, a condition defined in this document 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 (IV014) header field	implementation-defined (IV014)
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 (IV014) status field	implementation-defined (IV014)
L1: an implementation-defined (IL019) integer not less than 254.	

All diagnostics area fields in other parts of the ISO/IEC 9075 series that are not included in this table are not applicable to foreign-data wrapper interface routines.

4.19.7 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 document. This indicates that the caller wishes to prohibit the return of this information.
- In lieu of input arguments where specifically allowed by this document. 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 (HV009)*.

4.19.8 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)
DYNAMIC_FUNCTION_CODE	INTEGER
KEY_TYPE	SMALLINT
TOP_LEVEL_COUNT	SMALLINT
implementation-defined (IV015) foreign-data wrapper descriptor header field	implementation-defined (IV015)
CARDINALITY	INTEGER
CHARACTER_SET_CATALOG	CHARACTER VARYING(L)
CHARACTER_SET_NAME	CHARACTER VARYING(L)
CHARACTER_SET_SCHEMA	CHARACTER VARYING(L)
COLLATION_CATALOG	CHARACTER VARYING(L)
COLLATION_NAME	CHARACTER VARYING(L)
COLLATION_SCHEMA	CHARACTER VARYING(L)
CURRENT_TRANSFORM_GROUP	CHARACTER VARYING(L1)
DATA	ANY

Field	Data Type
DATA_POINTER	host variable address
DATETIME_INTERVAL_CODE	SMALLINT
DATETIME_INTERVAL_PRECISION	SMALLINT
DEGREE	INTEGER
INDICATOR	INTEGER
KEY_MEMBER	SMALLINT
LENGTH	INTEGER
LEVEL	INTEGER
NAME	CHARACTER VARYING(L)
NULLABLE	SMALLINT
OCTET_LENGTH	INTEGER
PARAMETER_MODE	SMALLINT
PARAMETER_ORDINAL_POSITION	SMALLINT
PARAMETER_SPECIFIC_CATALOG	CHARACTER VARYING(L)
PARAMETER_SPECIFIC_NAME	CHARACTER VARYING(L)
PARAMETER_SPECIFIC_SCHEMA	CHARACTER VARYING(L)
PRECISION	SMALLINT
RETURNED_CARDINALITY	INTEGER
RETURNED_OCTET_LENGTH	INTEGER
SCALE	SMALLINT
SCOPE_CATALOG	CHARACTER VARYING(L)
SCOPE_NAME	CHARACTER VARYING(L)
SCOPE_SCHEMA	CHARACTER VARYING(L)
SPECIFIC_TYPE_CATALOG	CHARACTER VARYING(L)
SPECIFIC_TYPE_NAME	CHARACTER VARYING(L)
SPECIFIC_TYPE_SCHEMA	CHARACTER VARYING(L)
TYPE	SMALLINT
UNNAMED	SMALLINT

Field	Data Type
USER_DEFINED_TYPE_CATALOG	CHARACTER VARYING(<i>L</i>)
USER_DEFINED_TYPE_NAME	CHARACTER VARYING(<i>L</i>)
USER_DEFINED_TYPE_SCHEMA	CHARACTER VARYING(<i>L</i>)
implementation-defined (IV015) foreign-data wrapper descriptor item field	implementation-defined (IV015)
<i>L</i>: an implementation-defined (IL028) integer not less than 128 <i>L1</i>: the implementation-defined (IL028) maximum length for the <general value specification> CURRENT_TRANS- FORM_GROUP_FOR_TYPE.	

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

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 | DLURLCOMPLETEONLY
| DLURLCOMPLETEWRITE | DLURLPATH | DLURLPATHONLY | DLURLPATHWRITE
| 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

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 after the last 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 after the last 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 after the last SR: In an <option name>, the number of <identifier part>s shall be less than 128.

- 4) Insert after the last 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 after the last 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 ISO/IEC 9075-2.

Access Rules

No additional Access Rules.

General Rules

- 1) Insert after the last GR: A <foreign server name> identifies a foreign server.
- 2) Insert after the last GR: A <foreign-data wrapper name> identifies a foreign-data wrapper.
- 3) Insert after the last 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 after **the last SR**: DATALINK specifies the datalink type.
- 2) Insert after **the last SR**: If <data type> specifies DATALINK and <datalink control definition> is not specified, then NO LINK CONTROL is implicit.
- 3) Insert after **the last 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 27 — Valid combinations of <datalink file control option> resulting from this Syntax Rule are shown in Table 1, “Valid datalink file control options”.
- 4) Insert after **the last 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 after **the last 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.

6.1 <data type>

- 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) Insert after the last CR: Without Feature M001, “Datalinks”, conforming SQL language shall not contain a <datalink type>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

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) Augment SR 7) by adding “a new rightmost column” to the table in SR 7).

<i>SD</i>	<i>TD</i>
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
JS	N

- 3) Augment SR 7) by adding “a new row” to the table in SR 7).

Where: DL = Data link

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 6.29, “<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) Augment SR 2) by adding “<datalink value expression>” to the list of BNF non-terminals.

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 6.33, “<string value function>”, in ISO/IEC 9075-2.*

Function

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

Format

```
<character 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) **Augment SR 2)** by adding “<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>, and <url server expression>” to the list of immediately contained BNF non-terminal symbols.
- 2) **Insert after the last SR:** Let *DLCS* be the <character set name> of the datalink character set.
NOTE 28 — “datalink character set” is defined in *Subclause 4.18, “Datalinks”*.
- 3) **Insert after the last SR:** Let *DVE* be the <datalink value expression>.
- 4) **Insert after the last SR:** The declared type of <url complete expression> is variable-length character string with character set *DLCS* and an implementation-defined (*IL020*) maximum length.

6.4 <string value function>

- 5) Insert after the last SR: The declared type of <url complete for write expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL025) maximum length.
- 6) Insert after the last SR: The declared type of <url complete only expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL026) maximum length.
- 7) Insert after the last SR: The declared type of <url path expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL021) maximum length.
- 8) Insert after the last SR: The declared type of <url path for write expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL027) maximum length.
- 9) Insert after the last SR: The declared type of <url path only expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL022) maximum length.
- 10) Insert after the last SR: The declared type of <url scheme expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL023) maximum length.
- 11) Insert after the last SR: The declared type of <url server expression> is variable-length character string with character set *DLCS* and an implementation-defined (IL024) maximum length.

Access Rules

No additional Access Rules.

General Rules

- 1) Augment GR 2) by adding "<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>, and <url server expression>" to the list of immediately contained BNF non-terminal symbols.
- 2) Insert after the last 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 after the last 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 (UA072) manner.
 - b) Otherwise, the result is the File Reference of *DV*.
- 4) Insert after the last 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 (UA072) manner.
 - b) Otherwise, the result is the File Reference of *DV*.
- 5) Insert after the last GR: If <url complete only expression> is specified, then the result is the File Reference of *DV*.
- 6) Insert after the last 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 (UA072) 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 (UA072) 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 after the last 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 (UA072) 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 (UA072) 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 after the last 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 after the last 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 after the last 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 29 — “datalink character set” is defined in Subclause 4.18, “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. No further subrules of this General Rule are applied, and application of General Rules continues with the next General Rule.
 - b) If either DLNEWCOPY or DLPREVIOUSCOPY is specified, then:
 - i) Let *TIV* be 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 (22023)*.
 - iii) If *DLOC* is the null value, then an exception condition is raised: *data exception — null argument passed to datalink constructor (2201A)*.

- iv) If *TIV* is equal to 1 (one) and if the write token included in *DLOC* does not conform to implementation-defined (IE014) requirements, then an exception condition is raised: *datalink exception — invalid write token (HW004)*.
- 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 20.1, “URL format”, nor to an implementation-defined format, then an exception condition is raised: *data exception — invalid data specified for datalink (22017)*.
- 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 (2201D)*.
NOTE 30 — The term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”.
- f) 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 20.1, “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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

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) Insert before GR 2)c)i): If *T* is a foreign table with <table name> *FTN*, then the result of *TP* is effectively determined as follows:
 - a) 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*.
 - b) Case:
 - i) 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*.
 - 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 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 : FSConnectionHandle} 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 invocation of the Connect - Server () 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 QCH be the QueryContextHandle returned by invocation of the AllocQueryContext () in the library identified by $WRLN$ with $FSCH$ as input argument.
- e) Let $TEMP$ be either a <query specification> of the form “SELECT * FROM FTN ” or a <query specification> $IDQS$ of the form “SELECT *DistinctOrAll* $exp_1, exp_2, \dots, exp_n$ FROM $FTN_1, FTN_2, \dots, FTN_m$ WHERE BVE_1 AND BVE_2 AND ... AND BVE_p ”, where all of the following are true:
- i) *DistinctOrAll* is either “DISTINCT” or “ALL”.
 - ii) n , m , and p are implementation-dependent (UV035) numeric values.
 - iii) For all i , $1 \text{ (one)} \leq i \leq n$, exp_i is an implementation-dependent (UV035) <value expression> that does not generally contain a <query expression> or a <routine invocation> one of whose subject routines possibly reads SQL-data.
 - iv) For all i , $1 \text{ (one)} \leq i \leq m$, FTN_i is a foreign table, and at least one of FTN_i shall be equivalent to FTN .
 - v) For all i , $1 \text{ (one)} \leq i \leq m$, the table descriptor of FTN_i shall include a foreign server descriptor that is equal to FSN .
 - vi) For all i , $1 \text{ (one)} \leq i \leq p$, 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.
- f) Let RQH be the RequestHandle allocated for $TEMP$.
- g) Let NTR be the number of <table reference>s in $TEMP$. Let TRH_i , $1 \text{ (one)} \leq i \leq NTR$, be the TableReferenceHandle allocated for each <table reference> simply contained in $TEMP$.

- h) Let N be the number of <value expression>s contained in the <select list> simply contained in $TEMP$. Let CN_i , $1 \text{ (one)} \leq i \leq N$, be the i -th such <value expression>. Let VEH_i , $1 \text{ (one)} \leq i \leq N$ be the i -th ValueExpressionHandle allocated for CN_i .
- i) Let M be the number of <value expression>s contained in the <where clause> simply contained in $TEMP$. Let CN_i , $N+1 \leq i \leq N+M$, be the i -th such <value expression>. Let VEH_i , $N+1 \leq i \leq N+M$, be the i -th ValueExpressionHandle allocated for CN_i .
- j) For all i , $1 \text{ (one)} \leq i \leq N+M$, 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 .
- k) For all i , $1 \text{ (one)} \leq i \leq N+M$, if CN_i is a <column reference> of a table reference TRH_j , then VEH_i is associated with TRH_j .
- l) For each CN_i , $1 \text{ (one)} \leq i \leq N+M$, 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 (UV036). Let $VEDH_i$ be the ValueExpressionDescriptorHandle associated with the value expression descriptor allocated for CN_i . $VEDH_i$ is associated with VEH_i .
- m) A table reference descriptor TRD is automatically allocated. Each of the fields in TRD that have non-blank entries in Table 35, "Foreign-data wrapper descriptor field default values", is set to the specified default value. All other fields in TRD are initially undefined.
- n) The General Rules of Subclause 22.4, "Implicit DESCRIBE OUTPUT USING clause for external data", are applied with $TEMP$ as *SOURCE* and WRD as *DESCRIPTOR*.
- o) A wrapper parameter descriptor WPD is automatically allocated. Each of the fields in WPD that have non-blank entries in Table 35, "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 .
- p) The General Rules of Subclause 22.3, "Implicit DESCRIBE INPUT USING clause for external data", are applied with $TEMP$ as *SOURCE* and WPD as *DESCRIPTOR*.
- q) Let $TRDH$ be the TableReferenceDescriptorHandle allocated for TRD .
- r) 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.
- s) TRD and WPD are associated with EXH .
- t) Let $NRTR$ be the NumberOfTableReferences that would be returned by an invocation of GetNumReplyTableRefs() with RPH as the ReplyHandle parameter.
- u) Let TRN_i , $1 \text{ (one)} \leq i \leq NRTR$, be the TableReferenceNumber that would be returned by an invocation of GetReplyTableRef() with RPH as the ReplyHandle parameter and i as the Index parameter.
- v) Let $NSLE$ be the NumberOfSelectListElements that would be returned by an invocation of GetNumReplySelectElems() with RPH as the ReplyHandle parameter.

- w) Let $SELN_i$, $1 \text{ (one)} \leq i \leq NSLE$, be the SelectListElementNumber that would be returned by an invocation of `GetReplySelectElem()` with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.
- x) Let *NBVE* be the NumberOfBoolVEs that would be returned by an invocation of `GetNumReply-BoolVE()` with *RPH* as the ReplyHandle parameter.
- y) Let $BVEN_i$, $1 \text{ (one)} \leq i \leq NBVE$, be the BoolVENumber that would be returned by an invocation of `GetReplyBoolVE()` with *RPH* as the ReplyHandle parameter and *i* as the Index parameter.
- z) Let *RCA* be the ReplyCardinality that would be returned by an invocation of `GetReplyCardinality()` with *RPH* as the ReplyHandle parameter.
- aa) Let *REFC* be the ReplyExecFirstCost that would be returned by an invocation of `GetReply-FirstCost()` with *RPH* as the ReplyHandle parameter.
- ab) Let *RTEC* be the ReplyTotalExecCost that would be returned by an invocation of `GetReplyExecCost()` with *RPH* as the ReplyHandle parameter.
- ac) Let *RREC* be the ReplyReExecutionCost that would be returned by an invocation of `GetReplyReExecCost()` with *RPH* as the ReplyHandle parameter.
- ad) It is implementation-dependent (UA035) 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)t) through GR 1)ad) of this Subclause are applied with *RPHN* and *EXHN* as *RPH* and *EXH*, respectively.
- ae) The `FreeReplyHandle()` routine in the library identified by *WRLN* is invoked with *RPH* as the argument.
- af) It is implementation-dependent (UA024) whether GR 1)d) through GR 1)af) of this Subclause are applied once again.
- ag) 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 31, “Codes used for foreign-data wrapper descriptor fields”, as the FieldIdentifier parameter.
- ah) Let DT_j be the effective data type of the *j*-th column, for $1 \text{ (one)} \leq j \leq NC$, 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 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_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 31, “Codes used for foreign-data wrapper descriptor fields”, as the FieldIdentifier parameter.
- ai) Let *SRD* be the SRDHandle that would be returned by an invocation of `GetSRDHandle()` with *EXH* as the ExecutionHandle parameter.
- aj) Let TDT_j be the effective data type of the *j*-th <target specification>, for $1 \text{ (one)} \leq j \leq NC$, 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 Set - Descriptor () with SRD as the DescriptorHandle parameter, j as the RecordNumber parameter, and the code for the fields TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATE-TIME_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 31, “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 16, “Codes used for application data types in SQL/CLI”.

ak) For every DT_j and TDT_j , $1 \text{ (one)} \leq j \leq NC$:

- i) 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 (HV006)*.
- ii) 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 (HV006)*.
- iii) If DT_j is a row data type, then

Case:

- 1) If TDT_j is not a row data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
- 2) If TDT_j is a row data type and DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.

- iv) 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 18.5, “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 (HV006)*.
- 2) Otherwise, if DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.

- v) If DT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.

al) 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 31, “Codes used for foreign-data wrapper descriptor fields”, as the FieldIdentifier parameter.

- am) Let PDT_j be the effective data type of the j -th column, for $1 \text{ (one)} \leq j \leq NP$, 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 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_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 31, “Codes used for foreign-data wrapper descriptor fields”, as the FieldIdentifier parameter.
- an) Let SPD be the SPDHandle that would be returned by an invocation of `GetSPDHandle()` with EXH as the ExecutionHandle parameter.
- ao) Let SDT_j be the effective data type of the j -th <target specification>, for $1 \text{ (one)} \leq j \leq NP$, 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 31, “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 16, “Codes used for application data types in SQL/CLI”.
- ap) For every PDT_j and SDT_j , $1 \text{ (one)} \leq j \leq NP$:
- i) 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 (HV006)*.
 - ii) 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 (HV006)*.
 - iii) If PDT_j is a row data type, then
Case:
 - 1) If SDT_j is not a row data type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
 - 2) If SDT_j is a row data type and PDT_j and SDT_j do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
 - iv) 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 18.5, “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 SDT_j is not a character string type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
- 2) Otherwise, if PDT_j and SDT_j do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
- v) If DT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
- aq) For all VEH_i , $1 \text{ (one)} \leq i \leq N+M$, 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.
- ar) VEH_i , $1 \text{ (one)} \leq i \leq N+M$, is deallocated and all its resources are freed.
- as) TRH is deallocated and all its resources are freed.
- at) RQH is deallocated and all its resources are freed.
- au) The `Open()` routine in the library identified by $WRLN$ is invoked with EXH as the argument.
- av) 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**.
- aw) The `Close()` routine in the library identified by $WRLN$ is invoked with EXH as the argument.
- ax) The `FreeExecutionHandle()` routine in the library identified by $WRLN$ is invoked with EXH as the argument.

Conformance Rules

No additional Conformance Rules.

8 Additional common rules

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

8.1 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).

Subclause Signature

"Retrieval assignment" [General Rules] (
 Parameter: "TARGET",
 Parameter: "VALUE"
)

TARGET — a site that is the target of a retrieval assignment operation.

VALUE — a value that is to be assigned to TARGET.

Syntax Rules

- 1) Insert after the last 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) Replicate GR 1): Let *T* be the *TARGET* and let *V* be the *VALUE* in an application of the General Rules of this Subclause.
- 2) Insert after GR 7)x): If the declared type of *T* is DATALINK, then the value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

8.2 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 after [the last SR](#): If the declared type of *T* is DATALINK, then the declared type of *V* shall be DATALINK.

Access Rules

No additional Access Rules.

General Rules

- 1) Insert after [GR 3\)b\)xxiv](#): If the declared type of *T* is DATALINK, then the value of *T* is set to *V*.

Conformance Rules

No additional Conformance Rules.

8.3 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.

8.4 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 after the last 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

8.5 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.

8.6 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 after the last 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

8.7 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 after the last 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.

8.8 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 after [the last 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.

8.9 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 after the last 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.

9 Additional common elements

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

9.1 <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 31 — 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.

9.2 <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.

10 Schema definition and manipulation

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

10.1 <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.

10.2 <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) Insert after [SR 4](#)n): Foreign tables.

Access Rules

No additional Access Rules.

General Rules

- 1) Insert after [GR 1](#)): Let *T* be the <table name> included in the descriptor of any foreign table included in *S*. The following <drop foreign table statement> is effectively executed:

DROP FOREIGN TABLE *T* CASCADE

Conformance Rules

No additional Conformance Rules.

10.3 <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 alternatives from ISO/IEC 9075-2  
    [ <datalink control definition> ]
```

Syntax Rules

- 1) Augment SR 11)g)iii) by adding “<datalink control definition>” to the list of BNF non-terminal symbols that a <column option list> shall immediately contain.
- 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.

10.4 <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.

10.5 <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 after [the last 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.

10.6 <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 after [the last GR](#): *D* shall not specify DATALINK.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

10.7 <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 32 — 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 4.18, “Datalinks”.

NOTE 33 — “constituent” is defined in Subclause 4.3, “Columns, fields, and attributes”.

Conformance Rules

No additional Conformance Rules.

10.8 <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.

10.9 <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 after [the last 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.

10.10 <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) Augment SR 2)c)i by adding “datalink type” to the list of types that *SDT* is not.

Conformance Rules

No additional Conformance Rules.

10.11 <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.

10.12 <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.

10.13 <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.

10.14 <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.

10.15 <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) $\leq i \leq n$:
 - a) For all *j*, 1 (one) $\leq j \leq n$, if the <column name> contained in the *i*-th <basic column definition> is equivalent to the <column name> contained in the *j*-th <basic column definition>, then *i=j*.
 - b) If the <data type> contained in the *i*-th <basic column definition> specifies a <character string type> and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification> of the <schema definition> that created the schema *S* is implicit.
- 4) Let *FSN* be the <foreign server name>.
- 5) The catalog identified by the explicit or implicit catalog name of *FSN* shall include a foreign server descriptor whose foreign server name is equivalent to *FSN*.
- 6) If the <foreign table definition> is contained in a <schema definition> *SD*, then let *A* be the explicit or implicit <authorization identifier> of *SD*. Otherwise, let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *TN*.

Access Rules

- 1) If <foreign table definition> is contained in an SQL-client module, then the enabled authorization identifiers shall include *A*.
- 2) 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 (IE023).

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 , $1 \text{ (one)} \leq i \leq n$, the corresponding *i*-th column descriptor includes:
 - 1) The <column name> contained in BCD_i .
 - 2) An indication that the column name is not an implementation-dependent (UV037) 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 (ID001) nullability characteristic.
 - 6) The implementation-defined (ID001) <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 (ID002).
 - 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 34 — This document currently restricts foreign tables such that they are neither insertable-into nor updatable. It is possible that this restriction will be relaxed in future editions.

- 2) Let *T* be the table described by *FTD*. Let *m* be the number of column descriptors CD_i , $1 \text{ (one)} \leq i \leq m$, included in *FTD*. The row type of *T* consists of *m* fields F_i such that, for all *i*, $1 \text{ (one)} \leq i \leq m$, the

10.15 <foreign table definition>

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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

10.16 <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*.

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> is specified, any effect on *FTD*, apart from that on its generic options descriptor, is implementation-defined (IA135).
- 3) 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>.

10.17 <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 (UV037) 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 (ID003) nullability characteristic.
 - f) The implementation-defined (ID003) <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.

10.18 <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.

10.19 <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 35 — 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 36 — 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 37 — *CN* can 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

10.20 <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 38 — 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 39 — 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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

11 Access control

This Clause modifies *Clause 12, “Access control”*, in ISO/IEC 9075-2.

11.1 <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) Insert after [SR 4\)a](#): If *ON* specifies a <foreign server name> or a <foreign-data wrapper name>, then *AC* shall specify USAGE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.2 <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 12)b):** 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 12)p):** 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 12)q)** by adding "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 (2B000)*.
- 4) **Insert after the last 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 after the last GR:** For every abandoned foreign table descriptor *FT*, let *S1.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.

11.3 <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 (IE024).

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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

11.4 <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*.

Access Rules

- 1) The privileges necessary to execute <alter user mapping statement> are implementation-defined (IW023).

General Rules

None.

Conformance Rules

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

11.5 <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 (IW024).

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>.

12 SQL-client modules

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

12.1 <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.

12.2 <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) after

package SQLSTATE_CODES **is**

the code:

```
INVALID_FOREIGN_SERVER_SPECIFICATION_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "0X000";
PASS_THROUGH_SPECIFIC_CONDITION_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "0Y000";
PASS_THROUGH_SPECIFIC_CONDITION_INVALID_CURSOR_OPTION:
  constant SQLSTATE_TYPE := "0Y001";
PASS_THROUGH_SPECIFIC_CONDITION_INVALID_CURSOR_ALLOCATION:
  constant SQLSTATE_TYPE := "0Y002";
DATA_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "22000";
DATA_EXCEPTION_INVALID_DATA_SPECIFIED_FOR_DATALINK:
  constant SQLSTATE_TYPE := "22017";
DATA_EXCEPTION_NULL_ARGUMENT_PASSED_TO_DATALINK_CONSTRUCTOR:
  constant SQLSTATE_TYPE := "2201A";
DATA_EXCEPTION_DATALINK_VALUE_EXCEEDS_MAXIMUM_LENGTH:
  constant SQLSTATE_TYPE := "2201D";
FDW_SPECIFIC_CONDITION_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "HV000";
FDW_SPECIFIC_CONDITION_MEMORY_ALLOCATION_ERROR:
  constant SQLSTATE_TYPE := "HV001";
FDW_SPECIFIC_CONDITION_INVALID_DATA_TYPE:
  constant SQLSTATE_TYPE := "HV004";
FDW_SPECIFIC_CONDITION_COLUMN_NAME_NOT_FOUND:
  constant SQLSTATE_TYPE := "HV005";
FDW_SPECIFIC_CONDITION_INVALID_DATA_TYPE_DESCRIPTOR:
  constant SQLSTATE_TYPE := "HV006";
FDW_SPECIFIC_CONDITION_INVALID_COLUMN_NAME:
  constant SQLSTATE_TYPE := "HV007";
FDW_SPECIFIC_CONDITION_INVALID_USE_OF_NULL_POINTER:
  constant SQLSTATE_TYPE := "HV009";
FDW_SPECIFIC_CONDITION_INVALID_STRING_FORMAT:
  constant SQLSTATE_TYPE := "HV00A";
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_OPTION_NAME_NOT_FOUND:
  constant SQLSTATE_TYPE := "HV00J";
```

```

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";
FDW_SPECIFIC_CONDITION_NO_SCHEMAS:
  constant SQLSTATE_TYPE := "HV00P";
FDW_SPECIFIC_CONDITION_SCHEMA_NOT_FOUND:
  constant SQLSTATE_TYPE := "HV00Q";
FDW_SPECIFIC_CONDITION_TABLE_NOT_FOUND:
  constant SQLSTATE_TYPE := "HV00R";
FDW_SPECIFIC_CONDITION_FUNCTION_SEQUENCE_ERROR:
  constant SQLSTATE_TYPE := "HV010";
FDW_SPECIFIC_CONDITION_LIMIT_ON_NUMBER_OF_HANDLES_EXCEEDED:
  constant SQLSTATE_TYPE := "HV014";
FDW_SPECIFIC_CONDITION_INCONSISTENT_DESCRIPTOR_INFORMATION:
  constant SQLSTATE_TYPE := "HV021";
FDW_SPECIFIC_CONDITION_INVALID_ATTRIBUTE_VALUE:
  constant SQLSTATE_TYPE := "HV024";
FDW_SPECIFIC_CONDITION_INVALID_STRING_LENGTH_OR_BUFFER_LENGTH:
  constant SQLSTATE_TYPE := "HV090";
FDW_SPECIFIC_CONDITION_INVALID_DESCRIPTOR_FIELD_IDENTIFIER:
  constant SQLSTATE_TYPE := "HV091";
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_REFERENCED_FILE_DOES_NOT_EXIST:
  constant SQLSTATE_TYPE := "HW003";
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_NOT_VALID:
  constant SQLSTATE_TYPE := "HW007";
CLI_SPECIFIC_CONDITION_NO_SUBCLASS:
  constant SQLSTATE_TYPE := "HY000";
CLI_SPECIFIC_CONDITION_INVALID_ATTRIBUTE_IDENTIFIER:
  constant SQLSTATE_TYPE := "HY092";
CLI_SPECIFIC_CONDITION_INVALID_DATALINK_VALUE:
  constant SQLSTATE_TYPE := "HY093";

```

The text of the Ada library unit package Interfaces.SQL is also available from the ISO website as a “digital artifact”. See <https://standards.iso.org/iso-iec/9075/-9/ed-5/en/> to download digital artifacts for this document. To download the library unit package, select the file named **ISO_IEC_9075-9(E)_MED-Interfaces.SQL.ada**.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

12.3 <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.

12.4 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

Insert into Table 21, “Data type correspondences for Ada” the rows of Table 5, “Data type correspondences for Ada”.

Table 5 — Data type correspondences for Ada

SQL Data Type	Ada Data Type
DATALINK	SQL_STANDARD.CHAR, with P'LENGTH of <i>LD</i>
<i>LD</i>: The length of the Ada character type corresponding with SQL data type DATALINK, which is the smallest integer not less than the quotient of the division N/B <i>N</i>: the maximum datalink length (the term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”) <i>B</i>: the implementation-defined (IV001) number of octets contained in a character of the host language	

Insert into Table 22, “Data type correspondences for C” the rows of Table 6, “Data type correspondences for C”.

Table 6 — Data type correspondences for C

SQL Data Type	C Data Type
DATALINK	char, with length <i>LD</i>
<i>LD</i>: the length of the C character type corresponding with SQL data type DATALINK, which is the smallest integer not less than the quotient of the division N/B <i>N</i>: the maximum datalink length (the term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”) <i>B</i>: the implementation-defined (IV002) number of octets contained in a character of the host language	

Insert into Table 23, “Data type correspondences for COBOL” the rows of Table 7, “Data type correspondences for COBOL”.

Table 7 — Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
DATALINK	alphanumeric, with length <i>LD</i>
<i>LD</i>: the length of the COBOL character type corresponding with SQL data type DATALINK, which is the smallest integer not less than the quotient of the division N/B <i>N</i>: the maximum datalink length (the term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”) <i>B</i> is the implementation-defined (IV003) number of octets contained in a character of the host language	

Insert into Table 24, “Data type correspondences for Fortran” the rows of Table 8, “Data type correspondences for Fortran”.

Table 8 — Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
DATALINK	CHARACTER with length <i>LD</i>
<i>LD</i>: the length of the Fortran character type corresponding with SQL data type DATALINK, which is the smallest integer not less than the quotient of the division N/B <i>N</i>: the maximum datalink length (the term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”) <i>B</i> is the implementation-defined (IV004) number of octets contained in a character of the host language	

Insert into Table 25, “Data type correspondences for M” the rows of Table 9, “Data type correspondences for M”.

Table 9 — Data type correspondences for M

SQL Data Type	MUMPS Data Type
DATALINK	character

Insert into Table 26, “Data type correspondences for Pascal” the rows of Table 10, “Data type correspondences for Pascal”.

Table 10 — Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
DATALINK	PACKED ARRAY[1.. <i>LD</i>] OF CHAR
<i>LD</i>: the length of the Pascal character type corresponding with SQL data type DATALINK, which is the smallest integer not less than the quotient of the division N/B <i>N</i>: the maximum datalink length (the term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”) <i>B</i>: the implementation-defined (IV005) number of octets contained in a character of the host language	

Insert into Table 27, “Data type correspondences for PL/I” the rows of Table 11, “Data type correspondences for PL/I”.

Table 11 — Data type correspondences for PL/I

SQL Data Type	PL/I Data Type
DATALINK	CHARACTER VARYING(<i>LD</i>)
<i>LD</i>: the length of the PL/I character type corresponding with SQL data type DATALINK, which is the smallest integer not less than the quotient of the division N/B <i>N</i>: the maximum datalink length (the term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”) <i>B</i> is the implementation-defined (IV006) number of octets contained in a character of the host language	

Conformance Rules

No additional Conformance Rules.

13 Additional data manipulation rules

This Clause modifies [Clause 15](#), “Additional data manipulation rules”, in ISO/IEC 9075-2.

13.1 Effect of deleting rows from base tables

This Subclause modifies [Subclause 15.8](#), “Effect of deleting rows from base tables”, in ISO/IEC 9075-2.

Function

Specify the effect of deleting rows that have been marked for deletion from one or more base tables.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

- 1) **Insert after GR 8):** 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 40 — “constituent” is defined in [Subclause 4.3](#), “Columns, fields, and attributes”.

- 2) **Insert after GR 8):** 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 (HW001)*.
- b) If *EF* is linked, then *EF* is unlinked.

NOTE 41 — The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in [Subclause 4.18](#), “Datalinks”.

Conformance Rules

No additional Conformance Rules.

13.2 Effect of inserting tables into base tables

This Subclause modifies Subclause 15.11, “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 8):** 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 42 — “constituent” is defined in Subclause 4.3, “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 (HW003)*.
 - 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 (HW005)*.
 - ii) If *EF* is linked, then an exception condition is raised: *datalink exception — external file already linked (HW002)*.
 - 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.

13.2 Effect of inserting tables into base tables

- 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

13.3 Effect of replacing rows in base tables

This Subclause modifies [Subclause 15.14](#), “Effect of replacing rows in base tables”, in ISO/IEC 9075-2.

Function

Specify the effect of replacing rows that have been marked for replacement in one or more base tables.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

- 1) **Insert after [GR 10](#):** 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 43 — “constituent” is defined in [Subclause 4.3](#), “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 (HW001)*.
- ii) If *EF1* is linked, *EF1* is unlinked.

NOTE 44 — The effect of unlinking depends on the unlink control option, RESTORE or DELETE, included in the column descriptor of *C*, as specified in [Subclause 4.18](#), “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 (HW003)*.
- 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 (HW002)*.
- 2) If the Construction Indication of *DLCV2* is either NEWCOPY or PREVIOUSCOPY, then:

13.3 Effect of replacing rows in base tables

- 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 (HW004)*.
 - II) If the Write Token of *DLCV2* is not valid according to implementation-defined (IE025) rules, then an exception condition is raised: *datalink exception — invalid write token (HW004)*.
- 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 (HW006)*.
- 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 (HW007)*.
- 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.

14 Session management

This Clause modifies *Clause 19, "Session management"*, in ISO/IEC 9075-2.

14.1 <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 )
```

14.1 <set passthrough statement>

- 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 (0X000)*.
 - 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 (0X000)*.
 - iv) If the current privileges do not include USAGE privilege on *FS*, then an exception condition is raised: *invalid foreign server specification (0X000)*.
 - 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>.

15 Dynamic SQL

This Clause modifies *Clause 20, “Dynamic SQL”, in ISO/IEC 9075-2.*

15.1 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)s): TYPE indicates DATALINK.
- 2) Insert before SR 7)z): TYPE indicates DATALINK and *T* is specified by DATALINK.

Access Rules

No additional Access Rules.

General Rules

- 1) Insert into Table 30, “Codes used for SQL data types in Dynamic SQL” the rows of Table 12, “Codes used for SQL data types in Dynamic SQL”.

Table 12 — Codes used for SQL data types in Dynamic SQL

Data Type	Code
DATALINK	70

Conformance Rules

No additional Conformance Rules.

15.2 <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 : WrapperEnvHandle} 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 : FSConnectionHandle} 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 *Connect-Server ()* 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.

15.3 <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 ExecutionHandle 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.

15.4 <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 (33000)*.
 - 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 (01005)*.
 - 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 (01005)*.
 - 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 45 — The term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”.

Conformance Rules

No additional Conformance Rules.

15.5 <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 by 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 (07001)*.
 - 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 (07008)*.

- iv) If the first N item descriptor areas in DA are not valid as specified in Subclause 15.1, “Description of SQL descriptor areas”, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications (07001)*.
- 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 (07001)*.
 - 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, MULTISSET, or MULTISSET 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 (07001)*.
- g) For $1 \text{ (one)} \leq i \leq D$:
 - 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 ISO/IEC 9075-2, and there is an implementation-defined conversion from type *SDT* to type *TDT*, then that implementation-defined conversion is effectively performed, converting *SV* to type *TDT*, and the result is the value TV_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 ISO/IEC 9075-2, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation (07006)*.

II) If the <cast specification>

CAST (SV AS TDT)

does not conform to the General Rules of Subclause 6.13, "<cast specification>", in ISO/IEC 9075-2, then an exception condition is raised in accordance with the General Rules of Subclause 6.13, "<cast specification>", in ISO/IEC 9075-2.

III) The <cast specification>

CAST (SV AS TDT)

is effectively performed and the result is the value TV_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 34, "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 34, "Ability to set foreign-data wrapper descriptor fields", are set to implementation-dependent (UV038) 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 34, “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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

15.6 <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 (07002)*.
 - 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 (07008)*.

- iv) If the first N item descriptor areas in DA are not valid as specified in Subclause 15.1, “Description of SQL descriptor areas”, then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications (07002)*.
- 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 (07002)*.
- g) For $1 \text{ (one)} \leq i \leq D$:
 - 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 (07006)* 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 ISO/IEC 9075-2, and there is an implementation-defined conversion from type *SDT* to type *TDT*, then that implementation-defined conversion is effectively performed, converting *SV* to type *TDT*, and the result is the value TV_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 ISO/IEC 9075-2, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation (07006)*.
 - II) If the <cast specification>

CAST (*SV* AS *TDT*)

does not conform to the General Rules of Subclause 6.13, "<cast specification>", in ISO/IEC 9075-2, then an exception condition is raised in accordance with the General Rules of Subclause 6.13, "<cast specification>", in ISO/IEC 9075-2.
 - III) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed and the result is the value TV_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 General Rules of Subclause 8.1, "Retrieval assignment", are applied with TV_i as *VALUE* and the *i*-th <into argument> as *TARGET*.
- h) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

15.7 <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 ExecutionHandle associated with <SQL statement name>.
 - b) If a <parameter using clause> is specified, then the General Rules specified in [Subclause 15.5](#), “<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 15.6](#), “<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.

15.8 <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 (0Y001)*.
 - b) If <cursor scrollability> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option (0Y001)*.
 - c) If <cursor holdability> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option (0Y001)*.
 - d) If <cursor returnability> is specified, then an exception condition is raised: *pass-through specific condition — invalid cursor option (0Y001)*.
 - e) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

15.9 <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 (0Y002)*.

Conformance Rules

No additional Conformance Rules.

15.10 <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 (0Y002)*.

Conformance Rules

No additional Conformance Rules.

15.11 <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 ExecutionHandle associated with <SQL statement name>.
 - b) If an <input using clause> is specified, then the General Rules specified in *Subclause 15.5*, “<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.

15.12 <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 ExecutionHandle 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 specified in Subclause 15.6, "<output using clause>", for a <result using clause> in a <dynamic fetch statement> are applied.
 - d) No further General Rules of this Subclause are applied.

Conformance Rules

No additional Conformance Rules.

15.13 <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 ExecutionHandle 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.

16 Embedded SQL

This Clause modifies [Clause 21](#), “Embedded SQL”, in ISO/IEC 9075-2.

16.1 <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 46 — The term “maximum datalink length” is defined in [Subclause 4.18](#), “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>.

16.1 <embedded SQL Ada program>

- 2) Without Feature M011, “Datalinks via Ada”, conforming SQL language shall not contain an <Ada DATALINK variable>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

16.2 <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 47 — The term “maximum datalink length” is defined in *Subclause 4.18, “Datalinks”*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1) Insert after the last CR: Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain a <C DATALINK variable>.
- 2) Insert after the last CR: Without Feature M012, “Datalinks via C”, conforming SQL language shall not contain a <C DATALINK variable>.

16.3 <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 48 — The term “maximum datalink length” is defined in [Subclause 4.18, “Datalinks”](#).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1) Insert after [the last CR](#): Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain a <COBOL DATALINK variable>.
- 2) Insert after [the last CR](#): Without Feature M013, “Datalinks via COBOL ”, conforming SQL language shall not contain a <COBOL DATALINK variable>.

16.4 <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 49 — The term “maximum datalink length” is defined in [Subclause 4.18, “Datalinks”](#).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1) Insert after [the last CR](#): Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain a <Fortran DATALINK variable>.
- 2) Insert after [the last CR](#): Without Feature M014, “Datalinks via Fortran”, conforming SQL language shall not contain a <Fortran DATALINK variable>.

16.5 <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) Insert after the last CR: Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain a <MUMPS DATALINK variable>.
- 2) Insert after the last CR: Without Feature M015, “Datalinks via M ”, conforming SQL language shall not contain a <MUMPS DATALINK variable>.

16.6 <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 50 — The term “maximum datalink length” is defined in [Subclause 4.18, “Datalinks”](#).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1) Insert after [the last CR](#): Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain a <Pascal DATALINK variable>.
- 2) Insert after [the last CR](#): Without Feature M016, “Datalinks via Pascal”, conforming SQL language shall not contain a <Pascal DATALINK variable>.

16.7 <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 51 — The term “maximum datalink length” is defined in [Subclause 4.18, “Datalinks”](#).

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

- 1) Insert after [the last CR](#): Without Feature M003, “Datalinks via Embedded SQL”, conforming SQL language shall not contain a <PL/I DATALINK variable>.
- 2) Insert after [the last CR](#): Without Feature M017, “Datalinks via PL/I”, conforming SQL language shall not contain a <PL/I DATALINK variable>.

17 Diagnostics management

This Clause modifies *Clause 23, “Diagnostics management”*, in ISO/IEC 9075-2.

17.1 <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) Insert into [Table 39, “SQL-statement codes”](#) the rows of [Table 13, “SQL-statement codes”](#).

Table 13 — SQL-statement codes

SQL-statement	Identifier	Code
<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
<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

SQL-statement	Identifier	Code
<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	y^2

² An implementation-defined (IV016) negative number different from the value associated with any other SQL-statement.

Conformance Rules

No additional Conformance Rules.

18 Call-Level Interface specifications

This Clause modifies *Clause 6, “Call-Level Interface specifications”*, in ISO/IEC 9075-3.

18.1 <CLI routine>

This Subclause modifies *Subclause 6.1, “<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)

Insert into Table 4, “Abbreviated SQL/CLI generic names”

 the rows of Table 14, “Abbreviated SQL/CLI generic names”.

Table 14 — Abbreviated SQL/CLI generic names

Generic Name	Abbreviation
BuildDataLink	BDL
GetDataLinkAttr	GDL

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

18.2 Implicit DESCRIBE USING clause

This Subclause modifies Subclause 6.9, “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.

Conformance Rules

No additional Conformance Rules.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

18.3 Description of CLI item descriptor areas

This Subclause modifies Subclause 6.17, “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) Augment SR 7)c)iv) by adding “DATALINK” to the list of types indicated.
- 3) Insert after SR 12)c)ix): TYPE indicates DATALINK and at least one of the following is true:
 - a) NULL is true.
 - b) DEFERRED is true.
- 4) Augment SR 13)c)iv) by adding “DATALINK” to the list of types indicated.

Access Rules

No additional Access Rules.

General Rules

- 1) Insert into Table 6, “Codes used for implementation data types in SQL/CLI” the rows of Table 15, “Codes used for implementation data types in SQL/CLI”.

Table 15 — Codes used for implementation data types in SQL/CLI

Data Type	Code
DATALINK	70

- 2) Insert into Table 7, “Codes used for application data types in SQL/CLI” the rows of Table 16, “Codes used for application data types in SQL/CLI”.

Table 16 — Codes used for application data types in SQL/CLI

Data Type	Code
DATALINK	70

Conformance Rules

No additional Conformance Rules.

18.4 Other tables associated with CLI

This Subclause modifies Subclause 6.18, “Other tables associated with CLI”, in ISO/IEC 9075-3.

Insert into Table 27, “Codes used to identify SQL/CLI routines” the rows of Table 17, “Codes used to identify SQL/CLI routines”.

Table 17 — Codes used to identify SQL/CLI routines

Generic Name	Code
BuildDataLink	1029
GetDataLinkAttr	1034

Insert into Table 28, “Codes and data types for implementation information” the rows of Table 18, “Codes and data types for implementation information”.

Table 18 — Codes and data types for implementation information

Information Type	Code	Data Type
MAXIMUM DATALINK LENGTH	20004	INTEGER

Table 19 — 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 (IV017) datalink attribute	< 0

Insert into Table 19, “Data types of attributes” the rows of Table 20, “Data types of attributes”.

Table 20 — Data types of attributes

Attribute	Data type	Values
URL COMPLETE	CHARACTER VARYING(L)	Datalink complete URL
URL PATH	CHARACTER VARYING(L)	Datalink URL path
URL PATH ONLY	CHARACTER VARYING(L)	Datalink URL path only

Attribute	Data type	Values
URL SCHEME	CHARACTER VARYING(<i>L</i>)	Datalink URL scheme
URL SERVER	CHARACTER VARYING(<i>L</i>)	Datalink URL server
implementation-defined (IV017) datalink attribute	implementation-defined (IV017) data type	implementation-defined (IV017) value
<i>L</i>: an implementation-defined (IL029) integer not less than the maximum datalink length. (The term “maximum datalink length” is defined in Subclause 4.18, “Datalinks”.)		

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

18.5 SQL/CLI data type correspondences

This Subclause modifies Subclause 6.19, “SQL/CLI data type correspondences”, in ISO/IEC 9075-3.

Function

Specify the SQL/CLI data type correspondences for SQL data types and host language types associated with the required parameter mechanisms, as shown in Table 3, “Supported calling conventions of SQL/CLI routines by language”, in ISO/IEC 9075-3.

Tables

Insert into Table 39, “SQL/CLI data type correspondences for Ada” the rows of Table 21, “SQL/CLI data type correspondences for Ada”.

Table 21 — SQL/CLI data type correspondences for Ada

SQL Data Type	Ada Data Type
DATALINK	SQL_STANDARD.CHAR, with P'Length of <i>LD</i>
<i>LD</i> : the length of the Ada character type corresponding with SQL data type DATALINK, which is implementation-defined (IV007)	

Insert into Table 40, “SQL/CLI data type correspondences for C” the rows of Table 22, “SQL/CLI data type correspondences for C”.

Table 22 — SQL/CLI data type correspondences for C

SQL Data Type	C Data Type
DATALINK	char, with length <i>LD</i>
<i>LD</i> : the length of the C character type corresponding with SQL data type DATALINK, is implementation-defined (IV008)	

Insert into Table 41, “SQL/CLI data type correspondences for COBOL” the rows of Table 23, “SQL/CLI data type correspondences for COBOL”.

Table 23 — SQL/CLI data type correspondences for COBOL

SQL Data Type	COBOL Data Type
DATALINK	alphanumeric, with length <i>LD</i> / <i>></i>
<i>LD</i> : the length of the COBOL character type corresponding with SQL data type DATALINK, which is is implementation-defined (IV009)	

Insert into Table 42, “SQL/CLI data type correspondences for Fortran” the rows of Table 24, “SQL/CLI data type correspondences for Fortran”.

Table 24 — SQL/CLI data type correspondences for Fortran

SQL Data Type	Fortran Data Type
DATALINK	CHARACTER with length <i>LD</i>
<i>LD</i> : the length of the Fortran character type corresponding with SQL data type DATALINK, which is implementation-defined (IV010)	

Insert into Table 43, “SQL/CLI data type correspondences for M” the rows of Table 25, “SQL/CLI data type correspondences for M”.

Table 25 — SQL/CLI data type correspondences for M

SQL Data Type	MUMPS Data Type
DATALINK	character

Insert into Table 44, “SQL/CLI data type correspondences for Pascal” the rows of Table 26, “SQL/CLI data type correspondences for Pascal”.

Table 26 — SQL/CLI data type correspondences for Pascal

SQL Data Type	Pascal Data Type
DATALINK	PACKED ARRAY[1.. <i>LD</i>] OF CHAR
<i>LD</i> : the length of the Pascal character type corresponding with SQL data type DATALINK, which is implementation-defined (IV011)	

Insert into Table 45, “SQL/CLI data type correspondences for PL/I” the rows of Table 27, “SQL/CLI data type correspondences for PL/I”.

Table 27 — SQL/CLI data type correspondences for PL/I

SQL Data Type	PL/I Data Type
DATALINK	CHARACTER VARYING(<i>LD</i>)
<i>LD</i> : the length of the PL/I character type corresponding with SQL data type DATALINK, which is implementation-defined (IV012)	

19 SQL/CLI routines

This Clause modifies [Clause 7](#), “SQL/CLI routines”, in ISO/IEC 9075-3.

19.1 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 (IL006) maximum length of a variable-length character string and *L2* has a maximum value equal to the implementation-defined (IL018) maximum length of a datalink.

General Rules

- 1) Let *SH* be the value of StatementHandle.
NOTE 52 — *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 53 — *File Reference* is defined in [Subclause 4.18](#), “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 (HY093)*.
NOTE 54 — The term “maximum datalink length” is defined in [Subclause 4.18](#), “Datalinks”.
- 5) The General Rules of [Subclause 22.7](#), “Character string retrieval for external data”, are applied with DataLink as *TARGET*, *DL* as *VALUE*, BufferLength as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

Conformance Rules

- 1) Without Feature M002, “Datalinks via SQL/CLI”, conforming SQL language shall not contain BuildDataLink().

19.2 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 (IL018) maximum length of a datalink.

General Rules

- 1) Let *SH* be the value of StatementHandle.
NOTE 55 — *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 19, “Codes used for datalink attributes”, then an exception condition is raised: *CLI-specific condition — invalid attribute identifier (HY092)*.
- 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: *FDW-specific condition — invalid string length or buffer length (HV090)*.
- 6) Let *ML* be the implementation-defined (IL018) 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 (HY093)*.
- 8) Let *BL* be the value of BufferLength.
- 9) If *A* specifies an implementation-defined (IV017) datalink attribute, then
Case:
 - a) If the data type for the datalink attribute is specified as INTEGER in Table 20, “Data types of attributes”, then Value is set to the value of the implementation-defined (IV017) datalink attribute and no further General Rules of this Subclause are applied.
 - b) Otherwise:
 - i) Let *AV* be the value of the implementation-defined (IV017) datalink attribute.

- ii) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with Value as *TARGET*, AV as *VALUE*, BL as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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

- 11) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with Value as *TARGET*, AV as *VALUE*, BL as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

Conformance Rules

- 1) Without Feature M002, “Datalinks via SQL/CLI”, conforming SQL language shall not contain Get - DataLinkAttr().

19.3 GetInfo()

This Subclause modifies *Subclause 7.39, “GetInfo()”, in ISO/IEC 9075-3.*

Function

Get information about the SQL-implementation.

Definition

No additional Definition.

General Rules

- 1)

Insert into GR 10)a) , after the 7th list item:

— MAXIMUM DATALINK LENGTH

Conformance Rules

- 1)

Insert after the last CR:

 Without Feature M002, “Datalinks via SQL/CLI”, in conforming SQL language, the value of InfoType shall not indicate MAXIMUM DATALINK LENGTH.

20 URLs

20.1 URL format

Function

Specify the precise format of a URL within a datalink. The specification is a direct translation of the format of HTTP and FILE URLs specified in [RFC 3986](#), except that “localhost” has been omitted from the format of FILE URL. [RFC 2368](#), [RFC 3986](#), [RFC 3978](#) and other documents specify also 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> ::=
    @ !! U+0040

```

<exclamation point> ::=
! !! U+0021

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

21 Catalog manipulation

21.1 <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 (IV018).

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 (IE026).

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.

21.1 <foreign server definition>

- 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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

21.2 <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) 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*.

Conformance Rules

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

21.3 <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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

21.4 <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 (UV039) <library name> is implicit.

Access Rules

- 1) The privileges necessary to execute <foreign-data wrapper definition> are implementation-defined (IW027).

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>.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

21.5 <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) 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>.

Conformance Rules

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

21.6 <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>.

21.7 <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 (HV00P)*.
- 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 (HV00Q)*.
- 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 (HV00R)*.
- 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 , $1 \text{ (one)} \leq i \leq n$, 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 , $1 \text{ (one)} \leq i \leq n$, 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>.

21.8 <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 (IE027).

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>.

21.9 <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*.

Access Rules

- 1) The privileges necessary to execute <alter routine mapping statement> are implementation-defined (IW029).

General Rules

None.

Conformance Rules

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

21.10 <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 (IW030).

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>.

22 SQL/MED common specifications

22.1 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 MULTISSET, 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) *SDE₁* 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 18.5, “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*.

22.1 Description of foreign-data wrapper item descriptor areas

- 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:
- TYPE indicates ROW or is one of the code values in Table 16, “Codes used for application data types in SQL/CLI”.
 - Exactly one of the following is true:
 - TYPE indicates NUMERIC, and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.
 - TYPE indicates DECIMAL, and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.
 - 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.
 - TYPE indicates FLOAT, and PRECISION is a valid precision value for the FLOAT data type.
 - TYPE indicates DECFLOAT, and PRECISION is a valid precision value for the DECFLOAT data type.
 - 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.
 - TYPE indicates ARRAY LOCATOR or MULTISSET LOCATOR, there is exactly 1 (one) immediately subordinate descriptor area of *IDA*, and that item descriptor area is valid.
 - TYPE indicates an implementation-defined (IE002) 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 26, “Miscellaneous codes used in CLI”, in ISO/IEC 9075-3, then NULL is true for *IDA*. Otherwise, NULL is false for *IDA*.
- 10) *IDA* is *valid* if and only if:
- TYPE is one of the code values in Table 16, “Codes used for application data types in SQL/CLI”, or TYPE indicates ROW.
 - 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.
 - Exactly one of the following is true:
 - 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:
 - 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.

22.1 Description of foreign-data wrapper item descriptor areas

- 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 MULTISSET 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 (IE002) data type.
- d) Case:
 - i) If *HL1* and *HL2* are both pointer-supporting languages, then exactly 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 16, "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) 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.

22.1 Description of foreign-data wrapper item descriptor areas

- 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 MULTISSET 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 (IE002) data type.

General Rules

None.

Conformance Rules

None.

22.2 Implicit foreign-data wrapper cursor

Function

Declare and open a foreign-data wrapper cursor.

Subclause Signature

“Implicit foreign-data wrapper cursor” [General Rules] (
Parameter: “ALLOCATED FDW-EXECUTION”
)

ALLOCATED FDW-EXECUTION — an allocated FDW-execution descriptor.

General Rules

- 1) Let *AE* be the *ALLOCATED FDW-EXECUTION* in an application of the General Rules 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 (UV040).
 - 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_TRANS-

FORM_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 *CID*.
 - e) Cursor *CN* is effectively placed in the open state with *RSD* as its result set descriptor.
- 5) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

22.3 Implicit DESCRIBE INPUT USING clause for external data

Function

Populate a specified descriptor area with information about the input values required to execute a foreign server request.

Subclause Signature

“Implicit DESCRIBE INPUT USING clause for external data” [General Rules] (
 Parameter: “SOURCE”,
 Parameter: “DESCRIPTOR”
)

SOURCE — a <query specification> for which information about input values is required.

DESCRIPTOR — a wrapper parameter descriptor area to be populated with information about the input values required.

General Rules

- 1) Let *S* be the *SOURCE* and let *DESC* be the *DESCRIPTOR* in an application of the General 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 13, “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 57 — 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 (UV041) 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 6, “Codes used for implementation data types in SQL/CLI”, in ISO/IEC 9075-3, indicating the data type of the <dynamic parameter specification> or subordinate descriptor.
 - iii) NULLABLE is set to 1 (one).

NOTE 58 — 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 (UV042) 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 8, “Codes associated with datetime data types in SQL/CLI”, in ISO/IEC 9075-3, to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision> if either is applicable.
 - 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 9, “Codes associated with <interval qualifier> in SQL/CLI”, in ISO/IEC 9075-3, to indicate the specific <interval qualifier>, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and PRECISION is set to the <interval fractional seconds precision>, if applicable.

22.3 Implicit DESCRIBE INPUT USING clause for external data

- 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 reference-able 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.
- 5) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

22.4 Implicit DESCRIBE OUTPUT USING clause for external data

Function

Populate a specified descriptor area with information about the values returned by an execution of a foreign server request.

Subclause Signature

“Implicit DESCRIBE OUTPUT USING clause for external data” [General Rules] (
 Parameter: “SOURCE”,
 Parameter: “DESCRIPTOR”
)

SOURCE — a <query specification> for which information about output values is required.

DESCRIPTOR — a wrapper row descriptor area to be populated with information about the output values required.

General Rules

- 1) Let *S* be the *SOURCE* and let *DESC* be the *DESCRIPTOR* in an application of the General 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 13, “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) $\leq i \leq D$, 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).

22.4 Implicit DESCRIBE OUTPUT USING clause for external data

- 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 (UV041) 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 6, "Codes used for implementation data types in SQL/CLI", in ISO/IEC 9075-3, indicating the data type of the column or subordinate descriptor.
 - iii) Case:
 - 1) If the value of LEVEL is 0 (zero), then:
 - A) If the resulting column is possibly nullable, then NULLABLE is set to 1 (one); otherwise NULLABLE is set to 0 (zero).
 - B) If the column name is implementation-dependent, then NAME is set to the implementation-dependent (UV042) 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 (UV042) name of the field and UNNAMED is set to 1 (one).

22.4 Implicit DESCRIBE OUTPUT USING clause for external data

- 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 (UV042) 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 8, "Codes associated with datetime data types in SQL/CLI", in ISO/IEC 9075-3, to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision> if either is applicable.
 - 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 9, "Codes associated with <interval qualifier> in SQL/CLI", in ISO/IEC 9075-3, to indicate the specific <interval qualifier>, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and PRECISION is set to the <interval fractional seconds precision>, if applicable.
 - 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.

22.4 Implicit DESCRIBE OUTPUT USING clause for external data

- 11) If TYPE indicates DATALINK, then LENGTH and OCTET_LENGTH are set to the maximum possible length in octets of the datalink.
- 5) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

22.5 Implicit EXECUTE USING and OPEN USING clauses for external data

Function

Specify the rules for an implicit EXECUTE USING clause and an implicit OPEN USING clause.

Subclause Signature

"Implicit EXECUTE USING and OPEN USING clauses for external data" [General Rules] (
 Parameter: "TYPE",
 Parameter: "ALLOCATED FDW-EXECUTION"
)

TYPE — the kind of execution that is intended: 'OPEN' or 'EXECUTE'.

ALLOCATED FDW-EXECUTION — an allocated FWD-execution descriptor.

General Rules

- 1) Let *T* be the *TYPE* and let *AE* be the *ALLOCATED FDW-EXECUTION* in an application of the General 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 (07008)*.
 - 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 22.1, "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 (07001)*.
 - 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 (07001)*.

22.5 Implicit EXECUTE USING and OPEN USING clauses for external data

- 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 (07001)*.
 - 1) The value of INDICATOR is not negative.
 - 2) Exactly one 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, MULTISSET, or MULTISSET 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.

- 5) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

22.6 Implicit FETCH USING clause for external data

Function

Specify the rules for an implicit FETCH USING clause.

Subclause Signature

“Implicit FETCH USING clause for external data” [General Rules] (
Parameter: “OPENED FDW-EXECUTION”
)

OPENED FDW-EXECUTION — an opened FDW-execution descriptor.

General Rules

- 1) Let *OE* be the *OPENED FDW-EXECUTION* in an application of the General 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 (07008)*.
 - 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 22.1, “Description of foreign-data wrapper item descriptor areas”, then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications (07002)*.
 - 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, CHARAC-

22.6 Implicit FETCH USING clause for external data

TER_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 6, "Codes used for implementation data types in SQL/CLI", in ISO/IEC 9075-3.
 - 2) Let *LV* be the implementation-defined (IL006) 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 (UV043) 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 (IL006) 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 26, “Miscellaneous codes used in CLI”, in ISO/IEC 9075-3 and the value of the host variable addressed by DP and the values of D and LP are implementation-dependent (UV044).
 - 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 26, “Miscellaneous codes used in CLI”, in ISO/IEC 9075-3, and the value of the host variable addressed by DP and the value of D and the value of LP are implementation-dependent (UV044).
 - 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 (IA087) whether or not an exception condition is raised: *data exception — zero-length character string (2200F)*.
 - 2) Case:
 - a) If $HL1$ and $HL2$ are both pointer-supporting languages, then the General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with DP as *TARGET*, TV as *VALUE*, OL as *TARGET OCTET LENGTH*, and LP as *RETURNED OCTET LENGTH*.
 - b) Otherwise, the General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with D as *TARGET*, TV as *VALUE*, OL as *TARGET OCTET LENGTH*, and LP as *RETURNED OCTET LENGTH*.
 - II) If TYPE indicates BINARY, BINARY VARYING, or BINARY LARGE OBJECT, then
 - Case:

22.6 Implicit FETCH USING clause for external data

- 1) If *HL1* and *HL2* are both pointer-supporting languages, then the General Rules of Subclause 22.8, “Binary string retrieval for external data”, are applied with *DP* as *TARGET*, *TV* as *VALUE*, *OL* as *TARGET OCTET LENGTH*, and *LP* as *RETURNED OCTET LENGTH*.
 - 2) Otherwise, the General Rules of Subclause 22.8, “Binary string retrieval for external data”, are applied with *D* as *TARGET*, *TV* as *VALUE*, *OL* as *TARGET OCTET LENGTH*, and *LP* as *RETURNED OCTET LENGTH*.
- 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 (UV044).
 - 2) Otherwise, the value of *D* is set to *TV* and the value of *LP* is implementation-dependent (UV044).
- 6) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

22.7 Character string retrieval for external data

Function

Specify the rules for retrieving character string values.

Subclause Signature

"Character string retrieval for external data" [General Rules] (
Parameter: "TARGET",
Parameter: "VALUE",
Parameter: "TARGET OCTET LENGTH",
Parameter: "RETURNED OCTET LENGTH"
)

TARGET — an identification of an SQL-data site where the result of character string retrieval is to be placed.

VALUE — an SQL-data value.

TARGET OCTET LENGTH — the length in octets of TARGET.

RETURNED OCTET LENGTH — a pointer to a host variable receiving the length in octets of VALUE, or a null pointer.

General Rules

- 1) Let *T* be the *TARGET*, let *V* be the *VALUE*, let *TL* be the *TARGET OCTET LENGTH*, and let *RL* be the *RETURNED OCTET LENGTH* in an application of the General Rules 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 (HV090)*.
- 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 (UV046).
 - b) Otherwise, *T* is set to the first *TL* octets of *V* and a completion condition is raised: *warning — string data, right truncation (01004)*.
- 6) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

22.8 Binary string retrieval for external data

Function

Specify the rules for retrieving binary string values.

Subclause Signature

"Binary string retrieval for external data" [General Rules] (
 Parameter: "TARGET",
 Parameter: "VALUE",
 Parameter: "TARGET OCTET LENGTH",
 Parameter: "RETURNED OCTET LENGTH"
)

TARGET — an identification of an SQL-data site where the result of binary string retrieval is to be placed.

VALUE — an SQL-data value.

TARGET OCTET LENGTH — the length in octets of TARGET.

RETURNED OCTET LENGTH — a pointer to a host variable receiving the length in octets of VALUE, or a null pointer.

General Rules

- 1) Let *T* be the *TARGET*, let *V* be the *VALUE*, let *TL* be the *TARGET OCTET LENGTH*, and let *RL* be the *RETURNED OCTET LENGTH* in an application of the General Rules 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 (HV090)*.
- 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 (UV046).
 - b) Otherwise, *T* is set to the first *TL* octets of *V* and a completion condition is raised: *warning — string data, right truncation (01004)*.
- 6) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

Conformance Rules

None.

22.9 Tables used with SQL/MED

The tables, such as Table 28, “Codes used for <table reference> types”, contained in this Subclause are used to specify the codes used by the various foreign-data wrapper interface routines.

Table 28 — Codes used for <table reference> types

<table reference> type	Code
TABLE_NAME	1

Table 29 — Codes used for <value expression> kinds

<value expression> kind	Code
COLUMN_NAME	1
CONSTANT	2
OPERATOR	3
PARAMETER	4

Table 30 — 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 (IV014) diagnostics header field	< 0	Header
implementation-defined (IV014) diagnostics status field	< 0	Status

Table 31 — 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
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

Field	Code	SQL Item Descriptor Name	Type
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
implementation-defined (IV015) foreign-data wrapper descriptor header field	0 (zero) through 999, or ≥ 1200 , excluding values defined in this table	implementation-defined (IV015) foreign-data wrapper descriptor header field	Header

Field	Code	SQL Item Descriptor Name	Type
implementation-defined (IV015) foreign-data wrapper descriptor item field	0 (zero) through 999, or ≥ 1200 , excluding values defined in this table	implementation-defined (IV015) foreign-data wrapper descriptor item field	Item

Table 32 — 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 33 — Ability to retrieve foreign-data wrapper descriptor fields

Field	May be retrieved			
	SRD	WRD or TRD	SPD	WPD
CARDINALITY	No		No	
CHARACTER_SET_CATALOG				
CHARACTER_SET_NAME				
CHARACTER_SET_SCHEMA				

	May be retrieved			
Field	SRD	WRD or TRD	SPD	WPD
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				
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

	May be retrieved			
Field	SRD	WRD or TRD	SPD	WPD
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				
implementation-defined (IV015) foreign-data wrapper descriptor header field	ID	ID	ID	ID
implementation-defined (IV015) foreign-data wrapper descriptor item field	ID	ID	ID	ID
“No”: the descriptor field is not retrievable PS: 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: the descriptor field is retrievable “ID”: it is implementation-defined (IA137) whether or not the descriptor field is retrievable				

Table 34 — Ability to set foreign-data wrapper descriptor fields

	May be set			
Field	SRD	WRD or TRD	SPD	WPD
CARDINALITY	No	No	No	

	May be set			
Field	SRD	WRD or TRD	SPD	WPD
CHARACTER_SET_CATALOG		No		
CHARACTER_SET_NAME		No		
CHARACTER_SET_SCHEMA		No		
COLLATION_CATALOG		No		
COLLATION_NAME		No		
COLLATION_SCHEMA		No		
COUNT		No		
CURRENT_TRANSFORM_GROUP	No	No	No	No
DATA		No		
DATA_POINTER		No		
DATETIME_INTERVAL_CODE		No		
DATETIME_INTERVAL_PRECISION		No		
DEGREE	No	No	No	
DYNAMIC_FUNCTION	No	No	No	No
DYNAMIC_FUNCTION_CODE	No	No	No	No
INDICATOR		No		No
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	

	May be set			
Field	SRD	WRD or TRD	SPD	WPD
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
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 (IV015) foreign-data wrapper descriptor header field	ID	ID	ID	ID
implementation-defined (IV015) foreign-data wrapper descriptor item field	ID	ID	ID	ID
“No”: the descriptor field is not settable “ID”: it is implementation-defined (IA138) whether or not the descriptor field is settable the absence of any notation means that the descriptor field is settable.				

Table 35 — 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)	
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				

	Default values			
Field	SRD	WRD or TRD	APD	WPD
PARAMETER_ORDINAL_POSITION				
PARAMETER_SPECIFIC_CATALOG				
PARAMETER_SPECIFIC_NAME				
PARAMETER_SPECIFIC_SCHEMA				
PRECISION				
RETURNED_CARDINALITY				
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 (IV015) foreign-data wrapper descriptor header field	ID	ID	ID	ID
implementation-defined (IV015) foreign-data wrapper descriptor item field	ID	ID	ID	ID
“Null”: the descriptor field’s default value is a null pointer the absence of any notation: the descriptor field’s default value is initially undefined “ID”: the descriptor field’s default value is implementation-defined (IW024) any other value: specifies the descriptor field’s default value				

Table 36 — Codes used for the format of the character string transmitted by GetSQLString()

Format	Code
SQL-string format	1
implementation-defined (IV019) formats	x
An implementation-defined (IV019) negative number different from the value associated with any other format	

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23 Foreign-data wrapper interface routines

23.1 <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

```

23.1 <foreign-data wrapper interface routine>

```

| 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>

```

23.1 <foreign-data wrapper interface routine>

```

    <foreign-data wrapper parameter mode>
    <foreign-data wrapper parameter data type>

<foreign-data wrapper parameter name> ::=
    !! See the Syntax Rules.

<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) A <foreign-data wrapper parameter name> shall be a parameter name in the Definition section of the Subclause in Clause 23, “Foreign-data wrapper interface routines”, whose name is the <foreign-data wrapper interface routine name> of the <foreign-data wrapper interface routine generic> of the containing <foreign-data wrapper interface routine>.
- 4) 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
```

- 5) *HL* shall support either the invocation of *WF* or the invocation of *WP*. It is implementation-defined (IA097) which is supported.
- 6) 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 59 — An output parameter is either a non-pointer host variable passed by reference or a pointer host variable passed by value.

- 7) 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>.

23.1 <foreign-data wrapper interface routine>

- 8) 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*.
- 9) Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 18.5, “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”.
- 10) 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 23.2, “<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 SQL-server 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 SQL-server 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 SQL-server shall not contain an invocation of a <foreign-data wrapper interface routine> written in PL/I.

23.2 <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 23.1, "<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 (IA114).
- 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:

23.2 <foreign-data wrapper interface routine> invocation

- 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 (00000)*, or a completion condition is raised: *no data (02000)*.
 - ii) Case:
 - 1) If a completion condition is raised: *successful completion (00000)*, then *RC* is set to indicate **Success**.
 - 2) If a completion condition is raised: *warning (01000)*, then *RC* is set to indicate **Success with information**.
 - 3) If a completion condition is raised: *no data (02000)*, 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 document or by implementation-defined (IA133) rules.
 - iii) Case:
 - 1) If an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*, 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 (UA036).

Conformance Rules

None.

23.3 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 (HV00B)*.
- 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 (HV00B)*.
- 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 (UW011) 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*.

23.4 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 (HV00B)*.
- 3) If the foreign-data wrapper implementation-defined (IL065) 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 (HV014)*. 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 (HV001)*.
NOTE 60 — 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 (IA171) reasons, then an implementation-defined (IC002) 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*.

23.5 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 (HV00B)*.
- 3) If the implementation-defined (IL030) 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 (HV014)*. 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 a 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 (HV001)*.

NOTE 61 — 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 a foreign-data wrapper environment cannot be allocated for implementation-defined (IA172) reasons, then an implementation-defined (IC003) 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 a 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 GetWrapperLibraryName() with *WH* as the WrapperHandle parameter.
- 8) It is implementation-dependent (UA037) 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.6 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 (HV010)*.
- 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 (24000)*.
 - 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.

23.7 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 (HV00B)*.
- 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 (HV00B)*.
- 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 (HV00B)*.
- 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 (IL031) 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 (HV014)*.
- 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 (HV001)*.
 - b) If the resources to manage an FS-connection cannot be allocated for implementation-defined (IA173) reasons, then FSConnectionHandle is set to zero and an implementation-defined (IC004) 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 (HV00N)*.
 - b) Otherwise, the connection to *FS* is established.
- 16) It is implementation-dependent (UA038) 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.

23.8 FreeExecutionHandle()

Function

Deallocate an FDW-execution.

Definition

FreeExecutionHandle (
 ExecutionHandle IN INTEGER)
 RETURNS SMALLINT

General Rules

- 1) Let *EH* be the value of ExecutionHandle.
- 2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
- 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`.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.9 FreeFSConnection()

Function

Deallocate a FS-connection.

Definition

FreeFSConnection (
 FSConnectionHandle IN INTEGER)
RETURNS SMALLINT

General Rules

- 1) Let *FSCH* be the value of FSConnectionHandle.
- 2) If *FSCH* does not identify an allocated FS-connection, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
- 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 (HV010)*.
- 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.

23.10 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 (HV00B)*.
- 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 (HV010)*.
- 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*.

23.11 FreeReplyHandle()

Function

Deallocate an FDW-reply.

Definition

FreeReplyHandle (IN INTEGER)
ReplyHandle
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 (HV00B)*.
- 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.

23.12 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 (HV00B)*.
- 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 (HV010)*.
- 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.

23.13 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 (HV00B)*.
- 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.

23.14 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 (HV00B)*.
- 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.

23.15 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 (HV00B)*.
- 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.

23.16 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 (HV00B)*.
- 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.

23.17 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 (HV00B)*.
- 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.

23.18 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 (IL006) 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 32, “Codes used for foreign-data wrapper handle types”, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
- 3) If *IH* does not identify a handle of the type indicated by *HT*, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
- 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 (02000)*.
 - 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 (02000)*.
 - 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 62 — The CDATA value of the SQLMEDOptionName attribute and the PCDATA text of the SQLMEDGenericOption tag are implementation-defined (IV020).

NOTE 63 — 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 (IA174).

- 5) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with Options as *TARGET*, *CD* as *VALUE*, *LO* as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.
- 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 64 — All negative values are reserved for use by foreign-data wrappers. All non-negative values are reserved for use by the ISO/IEC 9075 series.

Conformance Rules

- 1) Without Feature M009, “GetOpts 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.

23.19 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,
    BoolVNumber      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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, and no further rules of this Subclause are applied.
- 8) BoolVNumber 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.

23.21 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 (HV00B)*.
- 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.

23.22 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 (HV00B)*.
- 3) Let *C* be the estimated cost to retrieve the result set associated with *RH*.

NOTE 66 — 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.

23.23 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 (HV00B)*.
- 3) Let *C* be the estimated cost to retrieve the first row of the result set associated with *RH*.
NOTE 67 — 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.

23.24 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.25 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 (HV00B)*.
- 3) Let *C* be the estimated cost to re-execute the reply identified by *RH*.
NOTE 68 — 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.

23.26 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.27 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.28 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 (HV00B)*.
- 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.

23.29 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 (HV00B)*.
- 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.

23.30 GetStatistics()

Function

Retrieve implementation-defined (IV021) 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 (IL006) 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 (HV00B)*.
- 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 69 — The CDATA values of the *SQLMEDStatisticName* attribute and the PCDATA text of the *SQLMEDStatistics* tag are implementation-defined (IV022).

NOTE 70 — 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 (IA174).

- b) Otherwise, a completion condition is raised: *no data (02000)*.
- 4) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with *Statistics* as *TARGET*, *SI* as *VALUE*, *LOS* as *TARGET OCTET LENGTH*, and *StringLength* as *RETURNED OCTET LENGTH*.
- 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 71 — All negative values are reserved for use by foreign-data wrappers. All non-negative values are reserved for use by this document.

Conformance Rules

- 1) Without Feature M009, “GetOpts 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.31 GetWPDHandle()

Function

Get the descriptor handle of the wrapper parameter descriptor associated with a given execution handle.

Definition

GetWPDHandle (IN INTEGER,
 ExecutionHandle OUT INTEGER)
 WPDHandle
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 (HV00B)*.
- 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.

23.32 GetWRDHandle()

Function

Get the descriptor handle of the wrapper row descriptor associated with a given execution handle.

Definition

GetWRDHandle (IN INTEGER,
 ExecutionHandle OUT INTEGER)
 WRDHandle
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 (HV00B)*.
- 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.

23.33 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 (HV00B)*.
- 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 (UA039) whether *IG* is *True* or *False*.
NOTE 72 — 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) $\leq i \leq NTRE$:
 - 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 31,

“Codes used for foreign-data wrapper descriptor fields”, as the FieldIdentifier parameter.

- 4) For $1 \leq j \leq NC_i$, let DT_j 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 31, “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 $1 \leq k \leq NSLE$, 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 $1 \leq k \leq NBVE$, 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 $1 \leq m \leq NSLE+N BVE$, 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 GetValueExprColumnName() with VEH_m as the ValueExpressionHandle parameter.
- 7) If the implementation-defined (IL032) 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 (HV014)*.
- 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 (HV001)*.
 - b) If the resources to manage an FDW-reply cannot be allocated for implementation-defined (IA175) reasons, then ReplyHandle is set to zero and an implementation-defined (IC005) 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) $\leq i \leq NTRE$), (*DT_{ij}*, for 1 (one) $\leq i \leq NTRE$ and 1 (one) $\leq j \leq NC_i$), *NSLE*, and (*VEH_k*, *VET_k*, and *CN_k*, for 1 (one) $\leq k \leq NSLE+NBVE$), then an exception condition is raised: *FDW-specific condition — unable to create reply (HV00M)*.
- NOTE 73 — 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 (HV00M)*.
- c) Otherwise, the FDW-reply corresponding to *RH* is created.
- 10) If the implementation-defined (IL033) 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 (HV014)*.
- 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 (HV001)*.
- b) If the resources to manage an FDW-execution cannot be allocated for implementation-defined (IA176) reasons, then ExecutionHandle is set to zero and an implementation-defined (IC006) 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) $\leq i \leq NTRE$), (*DT_{ij}*, for 1 (one) $\leq i \leq NTRE$ and 1 (one) $\leq j \leq NC_i$), *NSLE*, and (*VEH_k*, *VET_k*, and *CN_k*, for 1 (one) $\leq k \leq NSLE+NBVE$), then an exception condition is raised: *FDW-specific condition — unable to create execution (HV00L)*.
- 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 (HV00L)*.
- 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 35, “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, $1 \text{ (one)} \leq r \leq NIDA$, and to the codes for the fields with non-blank entries in Table 35, “Foreign-data wrapper descriptor field default values”, from Table 31, “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 35, “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, $1 \text{ (one)} \leq r \leq NIDAP$, and to the codes for the fields with non-blank entries in Table 35, “Foreign-data wrapper descriptor field default values”, from Table 31, “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.

23.34 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 (HV010)*.
- 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 (02000)* 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 22.6, “Implicit FETCH USING clause for external data”, 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 (UV048) 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.

23.35 Open()

Function

Open an FDW-execution.

Definition

Open (
 ExecutionHandle IN INTEGER)
 RETURNS SMALLINT

General Rules

- 1) Let *EH* be the value of ExecutionHandle.
- 2) The General Rules of [Subclause 23.37, “Execution of Open\(\)”](#), are applied with *EH* as *EXECHANDLE*.

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.

23.36 ReOpen()

Function

Reopen an FDW-execution.

Definition

ReOpen (
 ExecutionHandle IN INTEGER)
RETURNS SMALLINT

General Rules

- 1) Let *EH* be the value of ExecutionHandle.
- 2) The General Rules of [Subclause 23.37, “Execution of Open\(\)”](#), are applied with *EH* as *EXECHANDLE*.

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.

23.37 Execution of Open()

Function

Execute an open or a reopen of an FDW-execution.

Subclause Signature

“Execution of Open()” [General Rules] (
Parameter: “EXECHandle”
)

EXECHandle — an FDW-execution descriptor.

General Rules

- 1) Let *EH* be the *EXECHandle* in an application of the General Rules of this Subclause.
- 2) If *EH* does not identify an allocated FDW-execution, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
- 3) If *EH* identifies an opened FDW-execution, then an exception condition is raised: *FDW-specific condition — function sequence error (HV010)*.
- 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 *SRCHandle* that would be returned by an invocation of the *GetSRCHandle()* routine with *EH* as the *ExecutionHandle* parameter. Let *SPD* be the *SPCHandle* that would be returned by an invocation of the *GetSPCHandle()* routine with *EH* as the *ExecutionHandle* parameter. Let *WRD* be the *WRCHandle* that would be returned by an invocation of the *GetWRCHandle()* routine with *EH* as the *ExecutionHandle* parameter. Let *WPD* be the *WPCHandle* that would be returned by an invocation of the *GetWPCHandle()* 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 31, “Codes used for foreign-data wrapper descriptor fields”, as the *FieldIdentifier* parameter.
 - c) Let *DT_j* be the effective data type of the *j*-th column, for 1 (one) $\leq j \leq NCR$, 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 31, “Codes used for foreign-data wrapper descriptor fields”, as the *FieldIdentifier* parameter. TYPE is one of the code values in Table 15, “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 $1 \text{ (one)} \leq j \leq NCR$, 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 31, “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 16, “Codes used for application data types in SQL/CLI”.
- e) For every DT_j and TDT_j , $1 \text{ (one)} \leq j \leq NCR$:
- 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 (HV006)*.
 - 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 (HV006)*.
 - 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 (HV006)*.
 - 2) If TDT_j is a row type and DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
 - 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 18.5, “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 (HV006)*.
 - 2) Otherwise, if DT_j and TDT_j do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
 - v) If DT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
- 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 31, “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 $1 \text{ (one)} \leq j \leq NCP$, 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 31, “Codes used for foreign-data wrapper descriptor fields”, as the FieldIdentifier parameter. TYPE is one of the code values in Table 15, “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 $1 \text{ (one)} \leq j \leq NCP$, 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 31, “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 16, “Codes used for application data types in SQL/CLI”.
- i) For every PDT_j and $PTDT_j$, $1 \text{ (one)} \leq j \leq NCP$:
 - 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 (HV006)*.
 - 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 (HV006)*.
 - 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 (HV006)*.
 - 2) If $PTDT_j$ is a row data type and PDT_j and $PTDT_j$ do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
 - 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 18.5,

“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 (HV006)*.
- 2) Otherwise, if PDT_j and $PTDT_j$ do not conform to the Syntax Rules of Subclause 9.30, “Data type identity”, in ISO/IEC 9075-2, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.
- v) If PDT_j is a user-defined type, then an exception condition is raised: *FDW-specific condition — invalid data type descriptors (HV006)*.

6) Case:

- a) If the foreign server request associated with EH returns a set of rows, then:
 - i) The General Rules of Subclause 22.5, “Implicit EXECUTE USING and OPEN USING clauses for external data”, are applied with OPEN as *TYPE* and S as *ALLOCATED FDW-EXECUTION*.
 - ii) The General Rules of Subclause 22.2, “Implicit foreign-data wrapper cursor”, are applied with S as *ALLOCATED FDW-EXECUTION*.
- b) Otherwise, the General Rules of Subclause 22.5, “Implicit EXECUTE USING and OPEN USING clauses for external data”, are applied with EXECUTE as *TYPE* and S as *ALLOCATED FDW-EXECUTION*.

7) EH is said to be an *opened FDW-execution*.

8) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause.

23.38 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 (IL006) 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 (HV00B)*.
- 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 (IL033) 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 (HV014)*.
- 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 (HV001)*.
 - b) If the resources to manage an FDW-execution cannot be allocated for implementation-defined (IA176) reasons, then *ReplyHandle* is set to zero and an implementation-defined (IC006) 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 (HV00M)*.
 - 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 35, “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, $1 \text{ (one)} \leq r \leq \textit{SRDItemDescriptorAreas}$, and the code for the fields with non-blank entries in Table 35, “Foreign-data wrapper descriptor field default values”, from Table 31, “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 35, “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, $1 \text{ (one)} \leq r \leq \textit{SPDItemDescriptorAreas}$, and the code for the fields with non-blank entries in Table 35, “Foreign-data wrapper descriptor field default values”, from Table 31, “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 35, “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, $1 \text{ (one)} \leq r \leq \textit{WRDItemDescriptorAreas}$, and the code for the fields with non-blank entries in Table 35, “Foreign-data wrapper descriptor field default values”, from Table 31, “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 35, “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, $1 \text{ (one)} \leq r \leq \textit{WPDItemDescriptorAreas}$, and the code for the fields with non-blank entries in Table 35, “Foreign-data wrapper descriptor field default values”, from Table 31, “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 22.4, “Implicit DESCRIBE OUTPUT USING clause for external data”, are applied with RequestString as *SOURCE* and *WRD* as *DESCRIPTOR*.

- 15) The General Rules of [Subclause 22.3](#), “Implicit DESCRIBE INPUT USING clause for external data”, are applied with RequestString as *SOURCE* and WPD as *DESCRIPTOR*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.39 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 (IL034) 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 (HV014)*.
- 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 (HV001)*.
 - b) If the resources to manage a foreign-data wrapper descriptor area cannot be allocated for implementation-defined (IA177) reasons, then DescriptorHandle is set to 0 (zero) and an implementation-defined (IC007) 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.

23.40 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 (HV00B)*.
- 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.

23.41 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 (IL005) 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 (HV00B)*.
- 3) Let *AID* be the authorization identifier associated with *UH*.
- 4) Let *BL* be the value of BufferLength.
- 5) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with AuthorizationId as *TARGET*, AID as *VALUE*, BL as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.42 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.43 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 31, “Codes used for foreign-data wrapper descriptor fields”, then an exception condition is raised: *FDW-specific condition — invalid descriptor field identifier (HV091)*.
- 4) Let *RN* be the value of *RecordNumber*.
- 5) Let *TYPE* be the value of the Type column in the row of Table 31, “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 (07009)*.
 - b) If *RN* is greater than *N*, then a completion condition is raised: *no data (02000)*.
- 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 (HV091)*.
- 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 74 — In the row of Table 33, “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 (IV027) descriptor header field, then the value retrieved is the value of the implementation-defined (IV027) descriptor header field identified by *FI*.

- c) Otherwise, if *FI* indicates a descriptor header field defined in Table 31, “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 75 — Let *MBR* be the value of the May Be Retrieved column in the row of Table 33, “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 (IV028) descriptor item field, then the value retrieved is the value of the implementation-defined (IV028) descriptor item field of *IDA* identified by *FI*.
 - b) Otherwise, if *FI* indicates a descriptor item field defined in Table 31, “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 22.7, “Character string retrieval for external data”, are applied with Value as *TARGET*, *V* as *VALUE*, *BL* as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.44 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 (HV00B)*.
- 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.

23.45 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 (HV00B)*.
- 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.

23.46 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 (HV00B)*.
- 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.

23.47 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 (HV00B)*.
- 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.

23.48 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 (HV00B)*.
- 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.

23.49 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 (HV00B)*.
- 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.

23.50 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 (HV00B)*.
- 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*.

23.51 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
- 5) Case:
 - a) If L is zero, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length (HV090)*.
 - 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 $NO \neq L$, then an exception condition is raised: *FDW-specific condition — invalid column name (HV007)*.
 - 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 (HV005)*.

- 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.52 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 (HV00B)*.
- 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.

23.53 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 (HV00B)*.
- 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.

23.54 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 (HV00B).
- 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.

23.55 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 (HV00B)*.
- 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.

23.56 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.57 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 (IL006) 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 (HV00B)*.
- 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 (HV00C)*.
- 6) If *ON* is greater than *N* then a completion condition is raised: *no data (02000)* 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 22.7, “Character string retrieval for external data”, are applied with OptionName as *TARGET*, NAME as *VALUE*, BufferLength1 as *TARGET OCTET LENGTH*, and StringLength1 as *RETURNED OCTET LENGTH*.
 - c) Let *OPTIONVALUE* be the value of the generic option.
 - d) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, *OPTIONVALUE* as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.58 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 (PL006) 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 (HV00B)*.
- 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 (HV090)*.
- 5) Case:
 - a) If *L* is zero, then an exception condition is raised: *FDW-specific condition — invalid string length or buffer length (HV090)*.
 - 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 (HV00D)*.
 - 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 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, OPTIONVALUE as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.
- b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found (HV00J)*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.59 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 (HV00B)*.
- 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.

23.60 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.61 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 (HV00B)*.
- 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.

23.62 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 $(2n+1)$, where n is the implementation-defined (IL006) length of an <identifier>.

NOTE 76 — The length $(2n+1)$ 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 (HV00B)*.
- 3) Let SN be the server name associated with SH .
- 4) Let BL be the value of BufferLength.
- 5) The General Rules of Subclause 22.7, "Character string retrieval for external data", are applied with ServerName as *TARGET*, SN as *VALUE*, BL as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.63 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 (IL006) 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 (HV00B)*.
- 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 (HV00C)*.
- 6) If *ON* is greater than *N*, then a completion condition is raised: *no data (02000)*, 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 22.7, “Character string retrieval for external data”, are applied with *OptionName* as *TARGET*, *NAME* as *VALUE*, *BufferLength1* as *TARGET OCTET LENGTH*, and *StringLength1* as *RETURNED OCTET LENGTH*.
 - c) Let *OPTIONVALUE* be the value of the generic option.
 - d) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with *OptionValue* as *TARGET*, *OPTIONVALUE* as *VALUE*, *BufferLength2* as *TARGET OCTET LENGTH*, and *StringLength2* as *RETURNED OCTET LENGTH*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.64 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
 - 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 (HV00D)*.
 - 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 22.7, “Character string retrieval for external data”, are applied with *OptionValue* as *TARGET*, *OPTIONVALUE* as *VALUE*, *BufferLength2* as *TARGET OCTET LENGTH*, and *StringLength2* as *RETURNED OCTET LENGTH*.

- b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found (HV00J)*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.65 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 (IL006) 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 (HV00B)*.
- 3) Let *ST* be the server type associated with *SH*.
- 4) Let *BL* be the value of *BufferLength*.
- 5) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with *ServerType* as *TARGET*, *ST* as *VALUE*, *BL* as *TARGET OCTET LENGTH*, and *StringLength* as *RETURNED OCTET LENGTH*.

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*.

23.66 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 (IL006) 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 (HV00B)*.
- 3) Let *SV* be the server version associated with *SH*.
- 4) Let *BL* be the value of *BufferLength*.
- 5) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with *ServerVersion* as *TARGET*, *SV* as *VALUE*, *BL* as *TARGET OCTET LENGTH*, and *StringLength* as *RETURNED OCTET LENGTH*.

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*.

23.67 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 (IL006) 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 (HV00B)*.
- 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 36, “Codes used for the format of the character string transmitted by GetSQLString()”, then an exception condition is raised: *FDW-specific condition — invalid string format (HV00A)*.
- 6) Case:
 - a) If *SF* identifies an implementation-defined (IV029) value, then let *QueryString* be the implementation-defined (IV029) character string representation of *Q*.
 - b) Otherwise, let *QueryString* be a character string conforming to the Format and Syntax Rules of Subclause 12.3, “<SQL procedure statement>”.
- 7) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with *SQLString* as *TARGET*, *QueryString* as *VALUE*, *BufferLength* as *TARGET OCTET LENGTH*, and *StringLength* as *RETURNED OCTET LENGTH*.

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*.

23.68 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
 - 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 $NO \neq L$, then an exception condition is raised: *FDW-specific condition — invalid column name (HV007)*.
 - 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 (HV005)*.
 - 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 (HV00C)*.
- iv) If ON is greater than N , then a completion condition is raised: *no data (02000)* 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 22.7, “Character string retrieval for external data”, are applied with OptionName as *TARGET*, NAME as *VALUE*, BufferLength1 as *TARGET OCTET LENGTH*, and StringLength1 as *RETURNED OCTET LENGTH*.
- 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 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, $VALUE$ as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

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 <foreign-data wrapper interface routine name> that contains GetTableColOpt.

23.69 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
 - 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 (HV007)*.
 - 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 (HV005)*.
 - 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 (HV090)*.
- 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 $BNO \neq BL1$, then an exception condition is raised: *FDW-specific condition — invalid option name (HV00D)*.
- 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 (HV00J)*.
- 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 22.7, “Character string retrieval for external data”, are applied with *OptionValue* as *TARGET*, *VALUE* as *VALUE*, *BufferLength2* as *TARGET OCTET LENGTH*, and *StringLength2* as *RETURNED OCTET LENGTH*.

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.

23.70 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 (IL006) 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 (HV00B)*.
- 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 (HV00C)*.
- 6) If *ON* is greater than *N* then a completion condition is raised: *no data (02000)* 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 22.7, “Character string retrieval for external data”, are applied with OptionName as *TARGET*, NAME as *VALUE*, BufferLength1 as *TARGET OCTET LENGTH*, and StringLength1 as *RETURNED OCTET LENGTH*.
- 9) Let *VALUE* be the value of the *ON*-th generic option associated with *TRH*.
- 10) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, VALUE as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

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.

23.71 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
 - 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 (HV00D)*.
 - 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 (HV00J)*.
 - 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 22.7](#), “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, VALUE as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.72 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 (HV00B)*.
- 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 (HV00C)*.
- 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 (02000)*, 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.

23.73 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 (HV00B)*.
- 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.

23.74 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 (IL006) 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 (HV00B)*.
- 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 (HV00B)*.
- 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 22.7, “Character string retrieval for external data”, are applied with TableName as *TARGET*, NAME as *VALUE*, BL as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.75 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 (IL005) 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 (HV00B)*.
- 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 22.7, “Character string retrieval for external data”, are applied with ServerName as *TARGET*, *SN* as *VALUE*, *BL* as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.76 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 (HV00B)*.
- 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.

23.77 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 (IL006) 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 (HV00B)*.
- 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 (HV00C)*.
- 6) If *ON* is greater than *N* then a completion condition is raised: *no data (02000)* 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 22.7, “Character string retrieval for external data”, are applied with OptionName as *TARGET*, NAME as *VALUE*, BufferLength1 as *TARGET OCTET LENGTH*, and StringLength1 as *RETURNED OCTET LENGTH*.
 - c) Let *OPTIONVALUE* be the value of the generic option.
 - d) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, *OPTIONVALUE* as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.78 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
 - 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 (HV00D)*.
 - 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 22.7, “Character string retrieval for external data”, are applied with *OptionValue* as *TARGET*, *OPTIONVALUE* as *VALUE*, *BufferLength2* as *TARGET OCTET LENGTH*, and *StringLength2* as *RETURNED OCTET LENGTH*.

- b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found (HV00J)*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.79 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 (IL006) 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 (HV00B)*.
- 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 (HV00B)*.
- 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 22.7, “Character string retrieval for external data”, are applied with ColumnName as *TARGET*, NAME as *VALUE*, BL as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.80 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 (HV00B)*.
- 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.

23.81 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 (HV00B)*.
- 3) Let *VEK* be the kind of the <value expression> associated with *VEH*.
NOTE 77 — The permissible values of the kind of a <value expression> are listed in Table 29, “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.

23.82 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 (IL006) 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 (HV00B)*.
- 3) Let *VEN* be the name associated with *VEH*.
- 4) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with ValueExpressionName as *TARGET*, *VEN* as *VALUE*, BufferLength as *TARGET OCTET LENGTH*, and ValueExpressionNameLength as *RETURNED OCTET LENGTH*.

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.

23.83 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 (HV00B)*.
- 3) Let *TBL* be the table associated with the allocated <value expression> identified by *VEH*. TableReferenceHandle 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.

23.84 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 (HV00B)*.
- 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 (HV00C)*.
- 6) If *I* is greater than *NOC*, then a completion condition is raised: *no data (02000)*, 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.

23.85 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 (IL006) 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 (HV00B)*.
- 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 22.7, “Character string retrieval for external data”, are applied with `WrapperLibraryName` as *TARGET*, WL as *VALUE*, `BufferLength` as *TARGET OCTET LENGTH*, and `StringLength` as *RETURNED OCTET LENGTH*.

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`.

23.86 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 $(2n+1)$, where n is the implementation-defined (IL005) length of an <identifier>.

NOTE 78 — The length $(2n+1)$ 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 (HV00B)*.
- 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 22.7 "Character string retrieval for external data", are applied with WrapperName as *TARGET*, WN as *VALUE*, BufferLength as *TARGET OCTET LENGTH*, and StringLength as *RETURNED OCTET LENGTH*.

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.

23.87 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 (IL006) 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 (HV00B)*.
- 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 (HV00C)*.
- 6) If *ON* is greater than *N* then a completion condition is raised: *no data (02000)* 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 22.7, “Character string retrieval for external data”, are applied with Optionname as *TARGET*, NAME as *VALUE*, BufferLength1 as *TARGET OCTET LENGTH*, and StringLength1 as *RETURNED OCTET LENGTH*.
 - c) Let *OPTIONVALUE* be the value of the generic option.
 - d) The General Rules of Subclause 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, *OPTIONVALUE* as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.88 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 (IL006) 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 (HV00B)*.
- 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 (HV090)*.
 - 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 (HV00D)*.
 - 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 22.7, “Character string retrieval for external data”, are applied with OptionValue as *TARGET*, *OPTIONVALUE* as *VALUE*, BufferLength2 as *TARGET OCTET LENGTH*, and StringLength2 as *RETURNED OCTET LENGTH*.

- b) Otherwise, an exception condition is raised: *FDW-specific condition — option name not found (HV00J)*.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.89 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 31, "Codes used for foreign-data wrapper descriptor fields", then an exception condition is raised: *FDW-specific condition — invalid descriptor field identifier (HV091)*.
- 5) Let *RN* be the value of *RecordNumber*.
- 6) Let *TYPE* be the value of the Type column in the row of Table 31, "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 (07009)*.
- 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 (UV049) values and the value of COUNT for *D* is unchanged.
- 10) Information is set in *D*.

NOTE 79 — Let *MBS* be the value of the May Be Set column in the row of Table 34, "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 (HV001)*.

- 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 (HV090)*.
 - 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 ISO/IEC 9075-2, 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 (01004)*.
 - 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 (3D000)*.
 - 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 (3F000)*.
 - 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 (2C000)*.
 - 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 (0700E)*.
 - 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, MULTISSET, or MULTISSET LOCATOR, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value (0700E)*.
 - ii) If *Value* is greater than *K*+1, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value (0700E)*.
 - 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 (0700E)*.
 - 2) If *K* is not 0 (zero), then let *OIDA_j* 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 (0700E)*.
 - 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_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 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 22.1, "Description of foreign-data wrapper item descriptor areas", then an exception condition is raised: *FDW-specific condition — inconsistent descriptor information (HV021)*.
- 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 (UV050) 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 (ID015) default value for the precision of the NUMERIC or the implementation-defined (ID016) default value for the precision of the 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 (IV128) 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 (IV128) 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 (IV128) 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 (IV128) value for the precision of the REAL or DOUBLE PRECISION data types, respectively.
 - x) If *V* indicates an implementation-defined (IV128) data type, then an implementation-defined (IV128) set of fields of *IDA* are set to implementation-defined (IV128) default values.
 - xi) Otherwise, an exception condition is raised: *FDW-specific condition — invalid data type (HV004)*.
- 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 (UV050) 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

23.90 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 32, “Codes used for foreign-data wrapper handle types”, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
- 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 (HV00B)*.
 - b) If *HT* indicates FS CONNECTION HANDLE and *Handle* does not identify an allocated FS-connection, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
 - 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 (HV00B)*.
 - 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 (HV00B)*.
 - 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 (HV00B)*.
 - f) If *HT* indicates TABLE REFERENCE HANDLE and *Handle* does not identify an allocated table reference description, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.
 - 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 (HV00B)*.
 - h) If *HT* indicates VALUE EXPRESSION HANDLE and *Handle* does not identify an allocated value expression description, then an exception condition is raised: *FDW-specific condition — invalid handle (HV00B)*.

- 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 (HV00B)*.
 - 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 (HV00B)*.
- 4) Let *DI* be the value of *DiagIdentifier*.
- 5) If *DI* is not one of the code values in Table 30, “Codes used for foreign-data wrapper diagnostic fields”, then an exception condition is raised: *FDW-specific condition — invalid attribute value (HV024)*.
- 6) Let *TYPE* be the value of the *Type* column in the row that contains *DI* in Table 30, “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 80 — The *GetDiagnostics()* routine can 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 (35000)*.
 - b) If *RN* is greater than *N*, then a completion condition is raised: *no data (02000)*, 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 RETURN CODE, then the value retrieved is the code indicating the basic result of the execution of *R*. Subclause 4.19.5, “Return codes”, specifies the code values and their meanings.
- NOTE 81 — 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 (IV014) diagnostics header field, then the value retrieved is the value of the implementation-defined (IV014) 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 (IV023) native error code corresponding to the status condition.
- c) If *DI* indicates `MESSAGE_TEXT`, then the value retrieved is an implementation-defined (IV024) character string.

NOTE 82 — An SQL-implementation can 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 (IV025) character string other than 'ISO 9075' for any implementation-defined (IV025) 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 (IV026) character string other than 'ISO 9075' for any implementation-defined (IV026) subclass code.
 - h) If *DI* indicates an implementation-defined (IV014) diagnostics status field, then the value retrieved is the value of the implementation-defined (IV014) 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 (HV090)*.
- 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 (UV051).
 - 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.

STANDARDSISO.COM : Click to view the full PDF of ISO/IEC 9075-9:2023

24 Information Schema

This Clause modifies Clause 6, “Information Schema”, in ISO/IEC 9075-11.

24.1 Information Schema digital artifact

This Subclause modifies Subclause 6.1, “Information Schema digital artifact”, in ISO/IEC 9075-11.

Insert after the 1st paragraph: These schema definition and manipulation statements are also available from the ISO website as a “digital artifact”. See <https://standards.iso.org/iso-iec/9075/-9/ed-5/en/> to download digital artifacts for this document. To download the schema definition and manipulation statements, select the file named `ISO_IEC_9075-9(E)_MED-schema-definition.sql`.

24.2 ATTRIBUTES view

This Subclause modifies Subclause 6.12, “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

Insert into the Definition after

MAXIMUM_CARDINALITY, A.DTD_IDENTIFIER, IS_DERIVED_REFERENCE_ATTRIBUTE,
DECLARED_DATA_TYPE, DECLARED_NUMERIC_PRECISION,
DECLARED_NUMERIC_SCALE

the code:

, 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 after the last CR:** Without Feature M001, “Datalinks”, conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.ATTRIBUTES:
 - a) DATALINK_INTEGRITY.
 - b) DATALINK_LINK_CONTROL.
 - c) DATALINK_READ_PERMISSION.
 - d) DATALINK_RECOVERY.
 - e) DATALINK_UNLINK.
 - f) DATALINK_WRITE_PERMISSION.

24.3 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 the view INFORMATION_SCHEMA.COLUMN_OPTIONS.

24.4 COLUMNS view

This Subclause modifies *Subclause 6.22, “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

Insert into the Definition after

```
COALESCE (D1.DECLARED_DATA_TYPE, D2.DECLARED_DATA_TYPE)
AS DECLARED_DATA_TYPE,
COALESCE (D1.DECLARED_NUMERIC_PRECISION, D2.DECLARED_NUMERIC_PRECISION)
AS DECLARED_NUMERIC_PRECISION,
COALESCE (D1.DECLARED_NUMERIC_SCALE, D2.DECLARED_NUMERIC_SCALE)
AS DECLARED_NUMERIC_SCALE
```

the code:

```
, DATALINK_LINK_CONTROL, DATALINK_INTEGRITY, DATALINK_READ_PERMISSION,
DATALINK_WRITE_PERMISSION, DATALINK_RECOVERY, DATALINK_UNLINK
```

Conformance Rules

- 1) Insert after the last CR: Without Feature M001, “Datalinks”, conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.COLUMNS:
 - a) DATALINK_INTEGRITY.
 - b) DATALINK_LINK_CONTROL.
 - c) DATALINK_READ_PERMISSION.
 - d) DATALINK_RECOVERY.
 - e) DATALINK_UNLINK.
 - f) DATALINK_WRITE_PERMISSION.

24.5 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 the view INFORMATION_SCHEMA . FOREIGN_DATA_WRAPPER_OPTIONS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA . FOREIGN_DATA_WRAPPER_OPTIONS.

24.6 FOREIGN_DATA_WRAPPERS view

Function

Identify the foreign-data wrappers that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_DATA_WRAPPERS AS
  SELECT FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
         AUTHORIZATION_IDENTIFIER, LIBRARY_NAME, FOREIGN_DATA_WRAPPER_LANGUAGE
  FROM DEFINITION_SCHEMA.FOREIGN_DATA_WRAPPERS W
 WHERE ( W.FOREIGN_DATA_WRAPPER_CATALOG, '', W.FOREIGN_DATA_WRAPPER_NAME )
        IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA, UP.OBJECT_NAME
              FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
              WHERE ( UP.GRANTEE IN
                      ( 'PUBLIC', CURRENT_USER )
                    OR
                      UP.GRANTEE IN
                      ( SELECT ROLE_NAME
                        FROM ENABLED_ROLES ) ) ) )
 AND
  W.FOREIGN_DATA_WRAPPER_CATALOG
  = ( SELECT CATALOG_NAME
      FROM INFORMATION_SCHEMA.CATALOG_NAME ) ;

GRANT SELECT ON TABLE FOREIGN_DATA_WRAPPERS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_DATA_WRAPPERS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_DATA_WRAPPERS.

24.7 FOREIGN_SERVER_OPTIONS view

Function

Identify the options specified for foreign servers that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_SERVER_OPTIONS AS
  SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
         OPTION_NAME, OPTION_VALUE
  FROM DEFINITION_SCHEMA.FOREIGN_SERVER_OPTIONS
 WHERE ( FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME )
        IN ( SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME
              FROM INFORMATION_SCHEMA.FOREIGN_SERVERS ) ;
```

```
GRANT SELECT ON TABLE FOREIGN_SERVER_OPTIONS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_SERVER_OPTIONS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_SERVER_OPTIONS.

24.8 FOREIGN_SERVERS view

Function

Identify the foreign servers defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_SERVERS AS
  SELECT FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME,
         FOREIGN_DATA_WRAPPER_CATALOG, FOREIGN_DATA_WRAPPER_NAME,
         FOREIGN_SERVER_TYPE, FOREIGN_SERVER_VERSION,
         AUTHORIZATION_IDENTIFIER
  FROM DEFINITION_SCHEMA.FOREIGN_SERVERS FS
 WHERE ( FS.FOREIGN_SERVER_CATALOG, '', FS.FOREIGN_SERVER_NAME )
    IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA, UP.OBJECT_NAME
        FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
        WHERE ( UP.GRANTEE IN
                ( 'PUBLIC', CURRENT_USER )
              OR
                UP.GRANTEE IN
                ( SELECT ROLE_NAME
                  FROM ENABLED_ROLES ) ) )
    AND
        FS.FOREIGN_SERVER_CATALOG
        = ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA.CATALOG_NAME );

GRANT SELECT ON TABLE FOREIGN_SERVERS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_SERVERS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_SERVERS.

24.9 FOREIGN_TABLE_OPTIONS view

Function

Identify the options specified for foreign tables that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_TABLE_OPTIONS AS
  SELECT FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME,
         OPTION_NAME, OPTION_VALUE
  FROM DEFINITION_SCHEMA.FOREIGN_TABLE_OPTIONS
 WHERE ( FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME ) IN
        ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
          FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
          WHERE GRANTEE IN
            ( 'PUBLIC', CURRENT_USER )
        UNION
          SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
          FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
          WHERE GRANTEE IN
            ( 'PUBLIC', CURRENT_USER ) )
 AND
  FOREIGN_TABLE_CATALOG
  = ( SELECT CATALOG_NAME
      FROM INFORMATION_SCHEMA.CATALOG_NAME ) ;

GRANT SELECT ON TABLE FOREIGN_TABLE_OPTIONS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_TABLE_OPTIONS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_TABLE_OPTIONS.

24.10 FOREIGN_TABLES view

Function

Identify the foreign tables that are defined in this catalog.

Definition

```
CREATE VIEW FOREIGN_TABLES AS
  SELECT FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME,
         FOREIGN_SERVER_CATALOG, FOREIGN_SERVER_NAME
  FROM DEFINITION_SCHEMA.FOREIGN_TABLES
 WHERE ( FOREIGN_TABLE_CATALOG, FOREIGN_TABLE_SCHEMA, FOREIGN_TABLE_NAME ) IN
        ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
          FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
          WHERE GRANTEE IN
            ( 'PUBLIC', CURRENT_USER )
        UNION
          SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
          FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
          WHERE GRANTEE IN
            ( 'PUBLIC', CURRENT_USER ) )
 AND
  FOREIGN_TABLE_CATALOG
  = ( SELECT CATALOG_NAME
      FROM INFORMATION_SCHEMA.CATALOG_NAME ) ;

GRANT SELECT ON TABLE FOREIGN_TABLES
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_TABLES.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.FOREIGN_TABLES.

24.11 ROUTINE_MAPPING_OPTIONS view

Function

Identify the options specified for routine mappings that are defined in this catalog.

Definition

```
CREATE VIEW ROUTINE_MAPPING_OPTIONS AS
  SELECT RMO.ROUTINE_MAPPING_NAME, RMO.OPTION_NAME, RMO.OPTION_VALUE
  FROM DEFINITION_SCHEMA.ROUTINE_MAPPING_OPTIONS AS RMO
  WHERE RMO.ROUTINE_MAPPING_NAME IN
    ( SELECT RM.ROUTINE_MAPPING_NAME
      FROM INFORMATION_SCHEMA.ROUTINE_MAPPINGS AS RM );
```

```
GRANT SELECT ON TABLE ROUTINE_MAPPING_OPTIONS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.ROUTINE_MAPPING_OPTIONS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.ROUTINE_MAPPING_OPTIONS.

24.12 ROUTINE_MAPPINGS view

Function

Identify the routine mappings that are defined in this catalog.

Definition

```
CREATE VIEW ROUTINE_MAPPINGS AS
  SELECT RM.ROUTINE_MAPPING_NAME, RM.SPECIFIC_CATALOG,
         RM.SPECIFIC_SCHEMA, RM.SPECIFIC_NAME,
         RM.FOREIGN_SERVER_CATALOG, RM.FOREIGN_SERVER_NAME
  FROM DEFINITION_SCHEMA.ROUTINE_MAPPINGS AS RM
 WHERE ( RM.FOREIGN_SERVER_CATALOG, RM.FOREIGN_SERVER_NAME ) IN
        ( SELECT FS.FOREIGN_SERVER_CATALOG, FS.FOREIGN_SERVER_NAME
          FROM INFORMATION_SCHEMA.FOREIGN_SERVERS AS FS );
```

```
GRANT SELECT ON TABLE ROUTINE_MAPPINGS
  TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

- 1) Without Feature F391, “Long identifiers”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.ROUTINE_MAPPINGS.
- 2) Without Feature M004, “Foreign data support”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.ROUTINE_MAPPINGS.