
**Information technology — Database
languages — SQL —**

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

*Technologies de l'information — Langages de base de données — SQL —
Partie 3: Interface de niveau d'appel (SQL/CLI)*

Reference number
ISO/IEC 9075-3:1999(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 1999

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

Page

Foreword	ix
Introduction	xi
1 Scope	1
2 Normative references	3
3 Definitions, notations, and conventions	5
3.1 Definitions	5
3.1.1 Definitions provided in Part 3	5
3.2 Notations	5
3.3 Conventions	5
3.3.1 Specification of routine definitions	5
3.3.2 Relationships to other parts of ISO/IEC 9075	6
3.3.2.1 Clause, Subclause, and Table relationships	6
3.4 Object identifier for Database Language SQL	13
4 Concepts	15
4.1 Introduction to SQL/CLI	15
4.2 Return codes	18
4.3 Diagnostics areas in SQL/CLI	19
4.3.1 Setting of ROW_NUMBER and COLUMN_NUMBER fields	22
4.4 Miscellaneous characteristics	22
4.4.1 Handles	22
4.4.2 Null terminated strings	23
4.4.3 Null pointers	23
4.4.4 Environment attributes	23
4.4.5 Connection attributes	24
4.4.6 Statement attributes	24
4.4.7 CLI descriptor areas	25
4.4.8 Obtaining diagnostics during multi-row fetch	26
4.5 Client-server operation	26

5	Call-Level Interface specifications	27
5.1	<CLI routine>	27
5.2	<CLI routine> invocation	34
5.3	Implicit set connection	37
5.4	Implicit cursor	38
5.5	Implicit DESCRIBE USING clause	40
5.6	Implicit EXECUTE USING and OPEN USING clauses	46
5.7	Implicit CALL USING clause	52
5.8	Implicit FETCH USING clause	56
5.9	Character string retrieval	62
5.10	Binary large object string retrieval	63
5.11	Deferred parameter check	64
5.12	CLI-specific status codes	65
5.13	Description of CLI item descriptor areas	67
5.14	Other tables associated with CLI	77
5.15	Data type correspondences	100
6	SQL/CLI routines	109
6.1	AllocConnect	109
6.2	AllocEnv	110
6.3	AllocHandle	111
6.4	AllocStmt	114
6.5	BindCol	115
6.6	BindParameter	117
6.7	Cancel	122
6.8	CloseCursor	124
6.9	ColAttribute	125
6.10	ColumnPrivileges	127
6.11	Columns	133
6.12	Connect	143
6.13	CopyDesc	147
6.14	DataSources	148
6.15	DescribeCol	150
6.16	Disconnect	152
6.17	EndTran	154
6.18	Error	159
6.19	ExecDirect	161
6.20	Execute	164
6.21	Fetch	166
6.22	FetchScroll	169
6.23	ForeignKeys	173
6.24	FreeConnect	186
6.25	FreeEnv	187
6.26	FreeHandle	188
6.27	FreeStmt	191
6.28	GetConnectAttr	193

6.29	GetCursorName	195
6.30	GetData	196
6.31	GetDescField	203
6.32	GetDescRec	205
6.33	GetDiagField	207
6.34	GetDiagRec	216
6.35	GetEnvAttr	218
6.36	GetFeatureInfo	220
6.37	GetFunctions	223
6.38	GetInfo	224
6.39	GetLength	232
6.40	GetParamData	234
6.41	GetPosition	240
6.42	GetSessionInfo	242
6.43	GetStmtAttr	244
6.44	GetSubString	247
6.45	GetTypeInfo	249
6.46	MoreResults	253
6.47	NextResult	254
6.48	NumResultCols	255
6.49	ParamData	256
6.50	Prepare	262
6.51	PrimaryKeys	264
6.52	PutData	269
6.53	RowCount	272
6.54	SetConnectAttr	273
6.55	SetCursorName	275
6.56	SetDescField	277
6.57	SetDescRec	282
6.58	SetEnvAttr	284
6.59	SetStmtAttr	286
6.60	SpecialColumns	290
6.61	StartTran	297
6.62	TablePrivileges	299
6.63	Tables	304
7	Definition Schema	311
7.1	SQL_IMPLEMENTATION_INFO base table	311
7.2	SQL_SIZING base table	313
7.3	SQL_LANGUAGES base table	315
8	Conformance	317
8.1	Conformance to SQL/CLI	317
8.2	Claims of conformance	317

8.3	Extensions and options	318
Annex A	Typical header files	319
A.1	C header file SQLCLI.H	319
A.2	COBOL library item SQLCLI	333
Annex B	Sample C programs	343
B.1	Create table, insert, select	343
B.2	Interactive Query	346
B.3	Providing long dynamic arguments at Execute time	350
Annex C	Implementation-defined elements	355
Annex D	Implementation-dependent elements	369
Annex E	Incompatibilities with ISO/IEC 9075-3:1995	375
Annex F	Deprecated features	377
Annex G	SQL feature and package taxonomy	379
Index		381

TABLES

Tables	Page
1 Clause, Subclause, and Table relationships	6
2 Fields in SQL/CLI diagnostics areas	20
3 Supported calling conventions of SQL/CLI routines by language	30
4 Abbreviated SQL/CLI generic names	30
5 SQLSTATE class and subclass values for SQL/CLI-specific conditions	65
6 Fields in SQL/CLI row and parameter descriptor areas	72
7 Codes used for implementation data types in SQL/CLI	74
8 Codes used for application data types in SQL/CLI	75
9 Codes associated with datetime data types in SQL/CLI	76
10 Codes associated with <interval qualifier> in SQL/CLI	76
11 Codes associated with <parameter mode> in SQL/CLI	76
12 Codes used for diagnostic fields	77
13 Codes used for handle types	78
14 Codes used for transaction termination	78
15 Codes used for environment attributes	79
16 Codes used for connection attributes	79
17 Codes used for statement attributes	79
18 Codes used for FreeStmt options	80
19 Data types of attributes	80
20 Codes used for descriptor fields	81
21 Ability to set SQL/CLI descriptor fields	83
22 Ability to retrieve SQL/CLI descriptor fields	85
23 SQL/CLI descriptor field default values	87
24 Codes used for fetch orientation	89
25 Multi-row fetch status codes	89
26 Miscellaneous codes used in CLI	90
27 Codes used to identify SQL/CLI routines	90
28 Codes and data types for implementation information	92
29 Codes and data types for session implementation information	94
30 Values for ALTER TABLE with GetInfo	94
31 Values for FETCH DIRECTION with GetInfo	95
32 Values for GETDATA EXTENSIONS with GetInfo	95
33 Values for OUTER JOIN CAPABILITIES with GetInfo	95
34 Values for SCROLL CONCURRENCY with GetInfo	95
35 Values for TRANSACTION ISOLATION OPTION with GetInfo and StartTran	95
36 Values for TRANSACTION ACCESS MODE with StartTran	96
37 Codes used for concise data types	96
38 Codes used with concise datetime data types in SQL/CLI	97

39	Codes used with concise interval data types in SQL/CLI	97
40	Concise codes used with datetime data types in SQL/CLI	98
41	Concise codes used with interval data types in SQL/CLI	98
42	Special parameter values	99
43	Column types and scopes used with SpecialColumns	99
44	Data type correspondences for Ada	100
45	Data type correspondences for C	102
46	Data type correspondences for COBOL	103
47	Data type correspondences for Fortran	105
48	Data type correspondences for MUMPS	106
49	Data type correspondences for Pascal	107
50	Data type correspondences for PL/I	108
51	SQL/CLI feature taxonomy	379

Foreword

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

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

International Standard ISO/IEC 9075-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This second edition of this part of ISO/IEC 9075 cancels and replaces the first edition, ISO/IEC 9075-3:1995.

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

- *Part 1: Framework (SQL/Framework)*
- *Part 2: Foundation (SQL/Foundation)*
- *Part 3: Call-Level Interface (SQL/CLI)*
- *Part 4: Persistent Stored Modules (SQL/PSM)*
- *Part 5: Host Language Bindings (SQL/Bindings)*

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

Introduction

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

- 1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 9075.
- 2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of ISO/IEC 9075, constitute provisions of this part of ISO/IEC 9075.
- 3) Clause 3, “Definitions, notations, and conventions”, defines the notations and conventions used in this part of ISO/IEC 9075.
- 4) Clause 4, “Concepts”, presents concepts used in the definition of the Call-Level Interface.
- 5) Clause 5, “Call-Level Interface specifications”, defines facilities for using SQL through a Call-Level Interface.
- 6) Clause 6, “SQL/CLI routines”, defines each of the routines that comprise the Call-Level Interface.
- 7) Clause 7, “Definition Schema”, specifies extensions to the Definition Schema required for support of the Call-Level Interface.
- 8) Clause 8, “Conformance”, defines the criteria for conformance to this part of ISO/IEC 9075.
- 9) Annex A, “Typical header files”, is an informative Annex. It provides examples of typical definition files for application programs using the SQL Call-Level Interface.
- 10) Annex B, “Sample C programs”, is an informative Annex. It provides examples of using the SQL Call-Level Interface from the C programming language.
- 11) Annex C, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-defined.
- 12) Annex D, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-dependent.
- 13) Annex E, “Incompatibilities with ISO/IEC 9075-3:1995”, is an informative Annex. It identifies incompatibilities with ISO/IEC 9075-3:1995.
- 14) Annex F, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intends will not appear in a future revised version of ISO/IEC 9075.
- 15) Annex G, “SQL feature and package taxonomy”, is an informative Annex. It contains a taxonomy of features of the SQL language that are specified in this part of ISO/IEC 9075.

In the text of this part of ISO/IEC 9075, Clauses begin a new odd-numbered page, and in Clause 5, “Call-Level Interface specifications”, through Clause 8, “Conformance”, Subclauses begin a new page. Any resulting blank space is not significant.

Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)

1 Scope

This part of ISO/IEC 9075 defines the structures and procedures that may be used to execute statements of the database language SQL from within an application written in a standard programming language in such a way that procedures used are independent of the SQL statements to be executed.

2 Normative references

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Definitions, notations, and conventions

3.1 Definitions

3.1.1 Definitions provided in Part 3

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

- a) **handle**: A CLI object returned by an SQL/CLI implementation when a CLI resource is allocated and used by an SQL/CLI application to reference that CLI resource.
- b) **inner table**: The second operand of a left outer join or the first operand of a right outer join.
- c) **pseudo-column**: A column in a table that is not part of the descriptor for that table. An example of such a pseudo-column is an implementation-defined row identifier.

3.2 Notations

Insert this paragraph The syntax notation used in this part of ISO/IEC 9075 is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form"). This version of BNF is fully described in Subclause 6.1, "Notation", of ISO/IEC 9075-1.

3.3 Conventions

Insert this paragraph Except as otherwise specified in this part of ISO/IEC 9075, the conventions used in this part of ISO/IEC 9075 are identical to those described in ISO/IEC 9075-1 and ISO/IEC 9075-2.

3.3.1 Specification of routine definitions

The routines in this part of ISO/IEC 9075 are specified in terms of:

- **Function**: A short statement of the purpose of the routine.
- **Definition**: The name of the routine and the names, modes, and data types of its parameters.
- **General Rules**: A specification of the run-time effect of the routine. Where more than one General Rule is used to specify the effect of a routine, the required effect is that which would be obtained by beginning with the first General Rule and applying the Rules in numeric sequence until a Rule is applied that specifies or implies a change in sequence or termination of the application of the Rules. Unless otherwise specified or implied by a specific Rule that is applied, application of General Rules terminates when the last in the sequence has been applied.

3.3.2 Relationships to other parts of ISO/IEC 9075

3.3.2.1 Clause, Subclause, and Table relationships

Table 1—Clause, Subclause, and Table relationships

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Clause 1, "Scope"	Clause 1, "Scope"	ISO/IEC 9075-2
Clause 2, "Normative references"	Clause 2, "Normative references"	ISO/IEC 9075-2
Clause 3, "Definitions, notations, and conventions"	Clause 3, "Definitions, notations, and conventions"	ISO/IEC 9075-2
Subclause 3.1, "Definitions"	Subclause 3.1, "Definitions"	ISO/IEC 9075-2
Subclause 3.1.1, "Definitions provided in Part 3"	Subclause 3.1.5, "Definitions provided in Part 2"	ISO/IEC 9075-2
Subclause 3.2, "Notations"	Subclause 3.2, "Notation"	ISO/IEC 9075-2
Subclause 3.3, "Conventions"	Subclause 3.3, "Conventions"	ISO/IEC 9075-2
Subclause 3.3.1, "Specification of routine definitions"	(none)	(none)
Subclause 3.3.2, "Relationships to other parts of ISO/IEC 9075"	(none)	(none)
Subclause 3.3.2.1, "Clause, Subclause, and Table relationships"	(none)	(none)
Subclause 3.4, "Object identifier for Database Language SQL"	none	none
Clause 4, "Concepts"	Clause 4, "Concepts"	ISO/IEC 9075-2
Subclause 4.1, "Introduction to SQL/CLI"	none	none
Subclause 4.2, "Return codes"	none	none
Subclause 4.3, "Diagnostics areas in SQL/CLI"	none	none
Subclause 4.3.1, "Setting of ROW_NUMBER and COLUMN_NUMBER fields"	none	none
Subclause 4.4, "Miscellaneous characteristics"	none	none
Subclause 4.4.1, "Handles"	none	none
Subclause 4.4.2, "Null terminated strings"	none	none
Subclause 4.4.3, "Null pointers"	none	none
Subclause 4.4.4, "Environment attributes"	none	none

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 4.4.5, "Connection attributes"	<i>none</i>	<i>none</i>
Subclause 4.4.6, "Statement attributes"	<i>none</i>	<i>none</i>
Subclause 4.4.7, "CLI descriptor areas"	<i>none</i>	<i>none</i>
Subclause 4.4.8, "Obtaining diagnostics during multi-row fetch"	<i>none</i>	<i>none</i>
Subclause 4.5, "Client-server operation"	Subclause 4.36, "Client-server operation"	ISO/IEC 9075-2
Clause 5, "Call-Level Interface specifications"	<i>none</i>	<i>none</i>
Subclause 5.1, "<CLI routine>"	<i>none</i>	<i>none</i>
Subclause 5.2, "<CLI routine> invocation"	<i>none</i>	<i>none</i>
Subclause 5.3, "Implicit set connection"	<i>none</i>	<i>none</i>
Subclause 5.4, "Implicit cursor"	<i>none</i>	<i>none</i>
Subclause 5.5, "Implicit DESCRIBE USING clause"	<i>none</i>	<i>none</i>
Subclause 5.6, "Implicit EXECUTE USING and OPEN USING clauses"	<i>none</i>	<i>none</i>
Subclause 5.7, "Implicit CALL USING clause"	<i>none</i>	<i>none</i>
Subclause 5.8, "Implicit FETCH USING clause"	<i>none</i>	<i>none</i>
Subclause 5.9, "Character string retrieval"	<i>none</i>	<i>none</i>
Subclause 5.10, "Binary large object string retrieval"	<i>none</i>	<i>none</i>
Subclause 5.11, "Deferred parameter check"	<i>none</i>	<i>none</i>
Subclause 5.12, "CLI-specific status codes"	<i>none</i>	<i>none</i>
Subclause 5.13, "Description of CLI item descriptor areas"	<i>none</i>	<i>none</i>
Subclause 5.14, "Other tables associated with CLI"	<i>none</i>	<i>none</i>
Subclause 5.15, "Data type correspondences"	Subclause 13.6, "Data type correspondences"	ISO/IEC 9075-2
Clause 6, "SQL/CLI routines"	<i>none</i>	<i>none</i>
Subclause 6.1, "AllocConnect"	<i>none</i>	<i>none</i>

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 6.2, “AllocEnv”	<i>none</i>	<i>none</i>
Subclause 6.3, “AllocHandle”	<i>none</i>	<i>none</i>
Subclause 6.4, “AllocStmt”	<i>none</i>	<i>none</i>
Subclause 6.5, “BindCol”	<i>none</i>	<i>none</i>
Subclause 6.6, “BindParameter”	<i>none</i>	<i>none</i>
Subclause 6.7, “Cancel”	<i>none</i>	<i>none</i>
Subclause 6.8, “CloseCursor”	<i>none</i>	<i>none</i>
Subclause 6.9, “ColAttribute”	<i>none</i>	<i>none</i>
Subclause 6.10, “ColumnPrivileges”	<i>none</i>	<i>none</i>
Subclause 6.11, “Columns”	<i>none</i>	<i>none</i>
Subclause 6.12, “Connect”	<i>none</i>	<i>none</i>
Subclause 6.13, “CopyDesc”	<i>none</i>	<i>none</i>
Subclause 6.14, “DataSources”	<i>none</i>	<i>none</i>
Subclause 6.15, “DescribeCol”	<i>none</i>	<i>none</i>
Subclause 6.16, “Disconnect”	<i>none</i>	<i>none</i>
Subclause 6.17, “EndTran”	<i>none</i>	<i>none</i>
Subclause 6.18, “Error”	<i>none</i>	<i>none</i>
Subclause 6.19, “ExecDirect”	<i>none</i>	<i>none</i>
Subclause 6.20, “Execute”	<i>none</i>	<i>none</i>
Subclause 6.21, “Fetch”	<i>none</i>	<i>none</i>
Subclause 6.22, “FetchScroll”	<i>none</i>	<i>none</i>
Subclause 6.23, “ForeignKeys”	<i>none</i>	<i>none</i>
Subclause 6.24, “FreeConnect”	<i>none</i>	<i>none</i>
Subclause 6.25, “FreeEnv”	<i>none</i>	<i>none</i>
Subclause 6.26, “FreeHandle”	<i>none</i>	<i>none</i>
Subclause 6.27, “FreeStmt”	<i>none</i>	<i>none</i>
Subclause 6.28, “GetConnectAttr”	<i>none</i>	<i>none</i>
Subclause 6.29, “GetCursorName”	<i>none</i>	<i>none</i>
Subclause 6.30, “GetData”	<i>none</i>	<i>none</i>
Subclause 6.31, “GetDescField”	<i>none</i>	<i>none</i>
Subclause 6.32, “GetDescRec”	<i>none</i>	<i>none</i>
Subclause 6.33, “GetDiagField”	<i>none</i>	<i>none</i>
Subclause 6.34, “GetDiagRec”	<i>none</i>	<i>none</i>

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 6.35, "GetEnvAttr"	<i>none</i>	<i>none</i>
Subclause 6.36, "GetFeatureInfo"	<i>none</i>	<i>none</i>
Subclause 6.37, "GetFunctions"	<i>none</i>	<i>none</i>
Subclause 6.38, "GetInfo"	<i>none</i>	<i>none</i>
Subclause 6.39, "GetLength"	<i>none</i>	<i>none</i>
Subclause 6.40, "GetParamData"	<i>none</i>	<i>none</i>
Subclause 6.41, "GetPosition"	<i>none</i>	<i>none</i>
Subclause 6.42, "GetSessionInfo"	<i>none</i>	<i>none</i>
Subclause 6.43, "GetStmtAttr"	<i>none</i>	<i>none</i>
Subclause 6.44, "GetSubString"	<i>none</i>	<i>none</i>
Subclause 6.45, "GetTypeInfo"	<i>none</i>	<i>none</i>
Subclause 6.46, "MoreResults"	<i>none</i>	<i>none</i>
Subclause 6.47, "NextResult"	<i>none</i>	<i>none</i>
Subclause 6.48, "NumResultCols"	<i>none</i>	<i>none</i>
Subclause 6.49, "ParamData"	<i>none</i>	<i>none</i>
Subclause 6.50, "Prepare"	<i>none</i>	<i>none</i>
Subclause 6.51, "PrimaryKeys"	<i>none</i>	<i>none</i>
Subclause 6.52, "PutData"	<i>none</i>	<i>none</i>
Subclause 6.53, "RowCount"	<i>none</i>	<i>none</i>
Subclause 6.54, "SetConnectAttr"	<i>none</i>	<i>none</i>
Subclause 6.55, "SetCursorName"	<i>none</i>	<i>none</i>
Subclause 6.56, "SetDescField"	<i>none</i>	<i>none</i>
Subclause 6.57, "SetDescRec"	<i>none</i>	<i>none</i>
Subclause 6.58, "SetEnvAttr"	<i>none</i>	<i>none</i>
Subclause 6.59, "SetStmtAttr"	<i>none</i>	<i>none</i>
Subclause 6.60, "SpecialColumns"	<i>none</i>	<i>none</i>
Subclause 6.61, "StartTran"	<i>none</i>	<i>none</i>
Subclause 6.62, "TablePrivileges"	<i>none</i>	<i>none</i>
Subclause 6.63, "Tables"	<i>none</i>	<i>none</i>
Clause 7, "Definition Schema"	Clause 21, "Definition Schema"	ISO/IEC 9075-2
Subclause 7.1, "SQL_IMPLEMENTATION_INFO base table"	Subclause 21.36, "SQL_IMPLEMENTATION_INFO base table"	ISO/IEC 9075-2
Subclause 7.2, "SQL_SIZING base table"	Subclause 21.38, "SQL_SIZING base table"	ISO/IEC 9075-2

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 7.3, "SQL_LANGUAGES base table"	Subclause 21.37, "SQL_LANGUAGES base table"	ISO/IEC 9075-2
Clause 8, "Conformance"	Clause 8, "Conformance"	ISO/IEC 9075-1
Subclause 8.1, "Conformance to SQL/CLI"	<i>none</i>	<i>none</i>
Subclause 8.2, "Claims of conformance"	Subclause 8.1.5, "Claims of conformance"	ISO/IEC 9075-1
Subclause 8.3, "Extensions and options"	<i>none</i>	<i>none</i>
Annex A, "Typical header files"	<i>none</i>	<i>none</i>
Subclause A.1, "C header file SQLCLI.H"	<i>none</i>	<i>none</i>
Subclause A.2, "COBOL library item SQLCLI"	<i>none</i>	<i>none</i>
Annex B, "Sample C programs"	<i>none</i>	<i>none</i>
Subclause B.1, "Create table, insert, select"	<i>none</i>	<i>none</i>
Subclause B.2, "Interactive Query"	<i>none</i>	<i>none</i>
Subclause B.3, "Providing long dynamic arguments at Execute time"	<i>none</i>	<i>none</i>
Annex C, "Implementation-defined elements"	Appendix B, "Implementation-defined elements"	ISO/IEC 9075-2
Annex D, "Implementation-dependent elements"	Appendix C, "Implementation-dependent elements"	ISO/IEC 9075-2
Annex E, "Incompatibilities with ISO/IEC 9075-3:1995"	Appendix E, "Incompatibilities with ISO/IEC 9075:1992 and ISO/IEC 9075-4:1996"	ISO/IEC 9075-2
Annex F, "Deprecated features"	Appendix D, "Deprecated features"	ISO/IEC 9075-2
Annex G, "SQL feature and package taxonomy"	<i>none</i>	<i>none</i>
Table 1, "Clause, Subclause, and Table relationships"	<i>none</i>	<i>none</i>
Table 2, "Fields in SQL/CLI diagnostics areas"	<i>none</i>	<i>none</i>
Table 3, "Supported calling conventions of SQL/CLI routines by language"	<i>none</i>	<i>none</i>
Table 4, "Abbreviated SQL/CLI generic names"	<i>none</i>	<i>none</i>
Table 5, "SQLSTATE class and subclass values for SQL/CLI-specific conditions"	<i>none</i>	<i>none</i>

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Table 6, “Fields in SQL/CLI row and parameter descriptor areas”	<i>none</i>	<i>none</i>
Table 7, “Codes used for implementation data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 8, “Codes used for application data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 9, “Codes associated with date-time data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 10, “Codes associated with <interval qualifier> in SQL/CLI”	<i>none</i>	<i>none</i>
Table 11, “Codes associated with <parameter mode> in SQL/CLI”	<i>none</i>	<i>none</i>
Table 12, “Codes used for diagnostic fields”	<i>none</i>	<i>none</i>
Table 13, “Codes used for handle types”	<i>none</i>	<i>none</i>
Table 14, “Codes used for transaction termination”	<i>none</i>	<i>none</i>
Table 15, “Codes used for environment attributes”	<i>none</i>	<i>none</i>
Table 16, “Codes used for connection attributes”	<i>none</i>	<i>none</i>
Table 17, “Codes used for statement attributes”	<i>none</i>	<i>none</i>
Table 18, “Codes used for FreeStmt options”	<i>none</i>	<i>none</i>
Table 19, “Data types of attributes”	<i>none</i>	<i>none</i>
Table 20, “Codes used for descriptor fields”	<i>none</i>	<i>none</i>
Table 21, “Ability to set SQL/CLI descriptor fields”	<i>none</i>	<i>none</i>
Table 22, “Ability to retrieve SQL/CLI descriptor fields”	<i>none</i>	<i>none</i>
Table 23, “SQL/CLI descriptor field default values”	<i>none</i>	<i>none</i>
Table 24, “Codes used for fetch orientation”	<i>none</i>	<i>none</i>
Table 25, “Multi-row fetch status codes”	<i>none</i>	<i>none</i>
Table 26, “Miscellaneous codes used in CLI”	<i>none</i>	<i>none</i>

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Table 27, “Codes used to identify SQL/CLI routines”	<i>none</i>	<i>none</i>
Table 28, “Codes and data types for implementation information”	<i>none</i>	<i>none</i>
Table 29, “Codes and data types for session implementation information”	<i>none</i>	<i>none</i>
Table 30, “Values for ALTER TABLE with GetInfo”	<i>none</i>	<i>none</i>
Table 31, “Values for FETCH DIRECTION with GetInfo”	<i>none</i>	<i>none</i>
Table 32, “Values for GETDATA EXTENSIONS with GetInfo”	<i>none</i>	<i>none</i>
Table 33, “Values for OUTER JOIN CAPABILITIES with GetInfo”	<i>none</i>	<i>none</i>
Table 34, “Values for SCROLL CONCURRENCY with GetInfo”	<i>none</i>	<i>none</i>
Table 35, “Values for TRANSACTION ISOLATION OPTION with GetInfo and StartTran”	<i>none</i>	<i>none</i>
Table 36, “Values for TRANSACTION ACCESS MODE with StartTran”	<i>none</i>	<i>none</i>
Table 37, “Codes used for concise data types”	<i>none</i>	<i>none</i>
Table 38, “Codes used with concise datetime data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 39, “Codes used with concise interval data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 40, “Concise codes used with datetime data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 41, “Concise codes used with interval data types in SQL/CLI”	<i>none</i>	<i>none</i>
Table 42, “Special parameter values”	<i>none</i>	<i>none</i>
Table 43, “Column types and scopes used with SpecialColumns”	<i>none</i>	<i>none</i>
Table 44, “Data type correspondences for Ada”	Table 18, "Data type correspondences for Ada"	ISO/IEC 9075-2
Table 45, “Data type correspondences for C”	Table 19, "Data type correspondences for C"	ISO/IEC 9075-2
Table 46, “Data type correspondences for COBOL”	Table 20, "Data type correspondences for COBOL"	ISO/IEC 9075-2

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

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Table 47, "Data type correspondences for Fortran"	Table 21, "Data type correspondences for Fortran"	ISO/IEC 9075-2
Table 48, "Data type correspondences for MUMPS"	Table 22, "Data type correspondences for MUMPS"	ISO/IEC 9075-2
Table 49, "Data type correspondences for Pascal"	Table 23, "Data type correspondences for Pascal"	ISO/IEC 9075-2
Table 50, "Data type correspondences for PL/I"	Table 24, "Data type correspondences for PL/I"	ISO/IEC 9075-2
Table 51, "SQL/CLI feature taxonomy"	Table 32, "SQL/Foundation feature taxonomy for features outside Core SQL"	ISO/IEC 9075-2

3.4 Object identifier for Database Language SQL

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

Format

<Part 3 yes> ::= <Part 3 conformance>

<Part 3 conformance> ::= 3 | sqlcli1999 <left paren> 3 <right paren>

Syntax Rules

None.

4 Concepts

4.1 Introduction to SQL/CLI

The Call-Level Interface (SQL/CLI) is a binding style for executing SQL statements. SQL/CLI comprises routines that:

- Allocate and deallocate resources.
- Control connections to SQL-servers.
- Execute SQL statements using mechanisms similar to dynamic SQL.
- Obtain diagnostic information.
- Control transaction termination.
- Obtain information about the implementation.

A *handle* is a CLI object returned by an SQL/CLI implementation when a CLI resource is allocated and used by an SQL/CLI application to reference that CLI resource. The `AllocHandle` routine allocates the resources to manage an SQL-environment, an SQL-connection, a CLI descriptor area, or SQL-statement processing and returns an environment handle, a connection handle, a descriptor handle, or a statement handle, respectively. An SQL-connection is allocated in the context of an allocated SQL-environment. A CLI descriptor area and an SQL-statement are allocated in the context of an allocated SQL-connection. The `FreeHandle` routine deallocates a specified resource. The `AllocConnect`, `AllocEnv`, and `AllocStmt` routines can be used to allocate the resources to manage an SQL-connection, an SQL-environment, and SQL-statement processing, respectively, instead of using the `AllocHandle` routine. The `FreeConnect`, `FreeEnv`, and `FreeStmt` routines can be used to deallocate the specific resource instead of using `FreeHandle`.

Each allocated SQL-environment has an attribute that determines whether output character strings are null terminated by the implementation. The application can set the value of this attribute by using the routine `SetEnvAttr` and can retrieve the current value of the attribute by using the routine `GetEnvAttr`.

The `Connect` routine establishes an SQL-connection. The `Disconnect` routine terminates an established SQL-connection. Switching between established SQL-connections occurs automatically whenever the application switches processing to a dormant SQL-connection.

The `ExecDirect` routine is used for a one-time execution of an SQL-statement. The `Prepare` routine is used to prepare an SQL-statement for subsequent execution using the `Execute` routine. In each case, the executed SQL-statement can contain dynamic parameters.

The interface for a description of dynamic parameters, dynamic parameter values, the resultant columns of a <dynamic select statement> or <dynamic single row select statement>, and the target specifications for the resultant columns is a CLI descriptor area. A CLI descriptor area for each type of interface is automatically allocated when an SQL-statement is allocated. The application may allocate additional CLI descriptor areas and nominate them for use as the interface for the description of dynamic parameter values or the description of target specifications by using the

routine SetStmtAttr. The application can determine the handle value of the CLI descriptor area currently being used for a specific interface by using the routine GetStmtAttr. The GetDescField and GetDescRec routines enable information to be retrieved from a CLI descriptor area. The CopyDesc routine enables the contents of a CLI descriptor area to be copied to another CLI descriptor area.

When a <dynamic select statement> or <dynamic single row select statement> is prepared or executed immediately, a description of the resultant columns is automatically provided in the applicable CLI implementation descriptor area. In this case, the application may additionally retrieve information by using the DescribeCol and/or the ColAttribute routine to obtain a description of a single resultant column and by using the NumResultCols routine to obtain a count of the number of resultant columns. The application sets values in the CLI application descriptor area for the description of the corresponding target specifications either explicitly using the routines SetDescField and SetDescRec or implicitly using the routine BindCol.

When an SQL-statement is prepared or executed immediately, a description of the dynamic parameters is automatically provided in the applicable CLI implementation descriptor area if this facility is supported by the current SQL-connection. An attribute associated with the allocated SQL-connection indicates whether this facility is supported. The value of the attribute may be retrieved using the routine GetConnectAttr. Regardless of whether automatic description is supported, all dynamic input and input/output parameters must be defined in the CLI application descriptor area before SQL-statement execution. This can be done either explicitly using the routines SetDescField and SetDescRec or implicitly using the routine BindParameter. The value of a dynamic input or input/output parameter may be established before SQL-statement execution (immediate parameter value) or may be provided during SQL-statement execution (deferred parameter value). Its description in the CLI descriptor area determines which method is in use. The ParamData routine is used to cycle through and process deferred input and input/output parameter values. The PutData routine is used to provide the deferred values. The PutData routine also enables the values of character string input and input/output parameters to be provided in pieces.

Before a <call statement> is prepared or executed immediately, the application may choose whether or not to bind any dynamic output parameters in the CLI application descriptor area. This can be done either explicitly using the routines SetDescField and SetDescRec or implicitly using the routine BindParameter. After execution of the statement, values of unbound output and input/output parameters can be individually retrieved using the GetParamData routine. The GetParamData routine also enables the retrieval of the values of character and binary string output and input/output parameters to be accomplished piece by piece.

When a <dynamic select statement> or <dynamic single row select statement> is executed, a cursor is implicitly declared and opened. The cursor name can be supplied by the application by using the routine SetCursorName. If a cursor name is not supplied by the application, an implementation-dependent cursor name is generated. The cursor name can be retrieved by using the GetCursorName routine.

The Fetch and FetchScroll routines are used to position an open cursor on a row and to retrieve the values of bound columns for that row. A bound column is one whose target specification in the specified CLI descriptor area defines a location for the target value. The Fetch routine always positions the open cursor on the next row, whereas the FetchScroll routine may be used to position the open cursor on any of its rows. The value of the CURSOR SCROLLABLE statement attribute must be SCROLLABLE at the time that the cursor is implicitly declared in order to use FetchScroll with a FetchOrientation other than NEXT. The application can set the value of this attribute by using the SetStmtAttr routine and can retrieve the current value of the attribute by using the GetStmtAttr routine. The Fetch and FetchScroll routines can also retrieve multiple rows in a single call; the set of rows thus retrieved is called a *rowset*. This is accomplished by setting the ARRAY_SIZE field of the applicable CLI application row descriptor to the desired number of rows. Note that the single row fetch is just a special case of multi-row fetch, where the rowset size is one.

Values for unbound columns can be individually retrieved by using the GetData routine. The GetData routine also enables the retrieval of the values of character and binary string columns to be accomplished piece by piece. The current row of a cursor can be deleted or updated by executing a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, respectively, for that cursor under a different allocated SQL-statement to the one under which the cursor was opened. The CloseCursor routine enables a cursor to be closed.

Result sets can be returned to the application as a result of issuing an Execute or ExecDirect routine against a statement handle whose current statement is a <call statement>. Such result sets are described and processed in the same way as outlined above for the result sets produced by the execution of a <dynamic select statement>. Multiple result sets may result from the execution of a single <call statement>. These result sets are returned as an ordered set of result sets that can be processed one at a time or in parallel. To process the result sets one at a time, once the processing of a given result set is complete, the MoreResults routine is used to determine whether there are any additional result sets and, if there are, to position the cursor before the first row in the next result set. To process the result sets in parallel, the NextResult routine is used to determine whether there are any additional result sets and, if there are, to position a cursor before the first row in the next result set.

Special routines, called *catalog routines* are available to return result sets from the information schemas. These routines are:

- ColumnPrivileges: Returns a list of the privileges held on the columns whose names adhere to the requested pattern(s) within a single specified table. Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the COLUMN_PRIVILEGES view of the Information Schema.
- Columns: Returns the column names and attributes for all columns that adhere to the requested pattern(s). Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the COLUMNS view of the Information Schema.
- ForeignKeys: Returns either the primary key of a single specified table together with the foreign keys in all other tables that reference that primary key or the foreign keys of a single specified table together with all the primary and unique keys in all other tables that reference those foreign keys. Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the TABLE_CONSTRAINTS view and the REFERENTIAL_CONSTRAINTS view of the Information Schema.
- PrimaryKeys: Returns a list of the columns that constitute the primary key of a single specified table. Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the TABLE_CONSTRAINTS view and the KEY_COLUMN_USAGE view of the Information Schema.
- SpecialColumns: Returns a list of the columns which can uniquely identify any row within a single specified table. Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the COLUMNS view of the Information Schema.
- Tables: Returns information about the tables that adhere to the requested pattern(s) and type(s). Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the TABLES view of the Information Schema.
- TablePrivileges: Returns a list of the privileges held on tables that adhere to the requested pattern(s). Most of this information can also be obtained by using the ExecDirect routine to issue an appropriate query on the TABLE_PRIVILEGES view of the Information Schema.

These special routines are only available for a small portion of the metadata that is available in the Information Schema. Other metadata (for example, that about SQL-invoked routines, triggers, and user-defined types) must be obtained by issuing appropriate queries on the views of the Information Schema.

The `GetPosition`, `GetLength`, and `GetSubString` routines can be used with their own independent statement handle to access a string value at the server that is represented by a Large Object locator in order to do any of the following:

- The `GetPosition` routine may be used determine whether a given substring exists within that string and, if it does, to obtain an integer value that indicates the starting position of that substring.
- The `GetLength` routine may be used to obtain an integer value that contains the length of the string.
- The `GetSubString` routine may be used to retrieve a portion of a string, or alternatively, to create a new Large Object value at the server which is a portion of the string and to return a Large Object locator that represents that value.

The `Error`, `GetDiagField`, and `GetDiagRec` routines obtain diagnostic information about the most recent routine operating on a particular resource. The `Error` routine always retrieves information from the next status record, whereas the `GetDiagField` and `GetDiagRec` routines may be used to retrieve information from any status record.

Information on the number of rows affected by the last executed SQL-statement can be obtained by using the `RowCount` or `GetDiagField` routine.

An SQL-transaction is terminated by using the `EndTran` routine. An SQL-transaction is implicitly initiated whenever a CLI routine is invoked that requires the context of an SQL-transaction and no SQL-transaction is active. An SQL-transaction is explicitly started and its characteristics set by using the `StartTran` routine.

NOTE 1 — Neither a <start transaction statement>, a <commit statement>, a <rollback statement>, nor a <release savepoint statement> may be executed using the `ExecDirect` or `Execute` routines.

The `Cancel` routine is used to cancel the execution of a concurrently executing SQL/CLI routine or to terminate the processing of deferred parameter values and the execution of the associated SQL-statement.

The `GetFeatureInfo`, `GetFunctions`, `GetInfo`, `GetSessionInfo`, and `GetTypeInfo` routines are used to obtain information about the implementation. The `DataSources` routine returns a list of names that identify SQL-servers to which the application may be able to connect and returns a description of each such SQL-server.

4.2 Return codes

The execution of a CLI routine causes one or more conditions to be raised. The status of the execution is indicated by a code that is returned either as the result of a CLI routine that is a CLI function or as the value of the `ReturnCode` argument of a CLI routine that is a CLI procedure.

The values and meanings of the return codes are as follows. If more than one return code is possible, then the one appearing later in the list is the one returned.

- A value of 0 (zero) indicates **Success**. The CLI routine executed successfully.

- A value of 1 (one) indicates **Success with information**. The CLI routine executed successfully but a completion condition was raised: *warning*.
- A value of 100 indicates **No data found**. The CLI routine executed successfully but a completion condition was raised: *no data*.
- A value of 99 indicates **Data needed**. The CLI routine did not complete its execution because additional data is needed. An exception condition was raised: *CLI-specific condition — dynamic parameter value needed*.
- A value of −1 indicates **Error**. The CLI routine did not execute successfully. An exception condition other than *CLI-specific condition — invalid handle* or *CLI-specific condition — dynamic parameter value needed* was raised.
- A value of −2 indicates **Invalid handle**. The CLI routine did not execute successfully because an exception condition was raised: *CLI-specific condition — invalid handle*.

After the execution of a CLI routine, the values of all output arguments not explicitly defined by this part of ISO/IEC 9075 are implementation-dependent.

In addition to providing the return code, for all CLI routines other than Error, GetDiagField, and GetDiagRec, the implementation records information about completion conditions and about exception conditions other than *CLI-specific condition — invalid handle* in the diagnostics area associated with the resource being utilized.

The resource being utilized by a routine is the resource identified by its input handle. In the case of CopyDesc, which has two input handles, the resource being utilized is deemed to be the one identified by TargetDescHandle.

4.3 Diagnostics areas in SQL/CLI

Each diagnostics area comprises header fields that contain general information relating to the routine that was executed and zero or more status records containing information about individual conditions that occurred during the execution of the CLI 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 CLI routine other than Error, GetDiagField, and GetDiagRec, the diagnostics area for the resource being utilized is emptied. If the execution of such a routine does not result in the exception condition *CLI-specific condition — invalid handle* or the exception condition *CLI-specific condition — dynamic parameter value needed*, 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 subclass value.

4.3 Diagnostics areas in SQL/CLI

When Fetch or FetchScroll is called to fetch a rowset and there are exceptions or warnings generated when fetching one or more rows, then the corresponding records in the diagnostics area have the ROW_NUMBER field set to the appropriate row number in the rowset. If a status record does not correspond to any row in the rowset, or the record is generated as a result of calling a routine other than Fetch or FetchScroll, the ROW_NUMBER field is set to zero. The COLUMN_NUMBER field of the diagnostic record contains the column number (if any) to which this diagnostic applies. If the diagnostic record does not apply to any column, then COLUMN_NUMBER is set to zero.

Status records in the diagnostics area are ordered by ROW_NUMBER. If multiple status records are generated for the same ROW_NUMBER value, then the order in which status records are placed in a diagnostics area is implementation-dependent except that:

- For the purpose of choosing the first status record, status records corresponding to *transaction rollback* have precedence over status records corresponding to other exceptions, which in turn have precedence over status records corresponding to the completion condition *no data*, which in turn have precedence over status records corresponding to the completion condition *warning*.
- Apart from any status records corresponding to an implementation-specified *no data*, any status record corresponding to an implementation-specified condition that duplicates, in whole or in part, a condition defined in this part of ISO/IEC 9075 shall not be the first status record.

The routines GetDiagField and GetDiagRec retrieve information from a diagnostics area. The application identifies which diagnostics area is to be accessed by providing the handle of the relevant resource as an input argument. The routines return a result code but do not modify the identified diagnostics area.

The Error routine also retrieves information from a diagnostics area. The Error routine retrieves the status records in the identified diagnostics area one at a time but does not permit already processed status records to be retrieved. Error returns a result code but does not modify the identified diagnostics area.

The RowCount routine retrieves the ROW_COUNT field from the diagnostics area for the specified statement handle. RowCount returns a result code and may cause exception or completion conditions to be raised which cause status records to be generated.

A CLI diagnostics area comprises the fields specified in Table 2, “Fields in SQL/CLI diagnostics areas”.

Table 2—Fields in SQL/CLI diagnostics areas

Field	Data type
Header fields	
DYNAMIC_FUNCTION	CHARACTER VARYING (<i>L1</i>)
DYNAMIC_FUNCTION_CODE	INTEGER
MORE	INTEGER
NUMBER	INTEGER
RETURNCODE	SMALLINT
ROW_COUNT	INTEGER
Where <i>L</i> is an implementation-defined integer not less than 128 and <i>L1</i> is an implementation-defined integer not less than 254.	

Table 2—Fields in SQL/CLI diagnostics areas (Cont.)

Field	Data type
Header fields	
TRANSACTIONS_COMMITTED	INTEGER
TRANSACTIONS_ROLLED_BACK	INTEGER
TRANSACTION_ACTIVE	INTEGER
Implementation-defined header field	Implementation-defined
Fields in status records	
CATALOG_NAME	CHARACTER VARYING (<i>L</i>)
CLASS_ORIGIN	CHARACTER VARYING (<i>L1</i>)
COLUMN_NAME	CHARACTER VARYING (<i>L</i>)
COLUMN_NUMBER	INTEGER
CONDITION_IDENTIFIER	CHARACTER VARYING (<i>L</i>)
CONDITION_NUMBER	INTEGER
CONNECTION_NAME	CHARACTER VARYING (<i>L</i>)
CONSTRAINT_CATALOG	CHARACTER VARYING (<i>L</i>)
CONSTRAINT_NAME	CHARACTER VARYING (<i>L</i>)
CONSTRAINT_SCHEMA	CHARACTER VARYING (<i>L</i>)
CURSOR_NAME	CHARACTER VARYING (<i>L</i>)
MESSAGE_LENGTH	INTEGER
MESSAGE_OCTET_LENGTH	INTEGER
MESSAGE_TEXT	CHARACTER VARYING (<i>L1</i>)
NATIVE_CODE	INTEGER
PARAMETER_MODE	CHARACTER VARYING (<i>L</i>)
PARAMETER_NAME	CHARACTER VARYING (<i>L</i>)
PARAMETER_ORDINAL_POSITION	INTEGER
ROUTINE_CATALOG	CHARACTER VARYING (<i>L</i>)
ROUTINE_NAME	CHARACTER VARYING (<i>L</i>)
ROUTINE_SCHEMA	CHARACTER VARYING (<i>L</i>)
ROW_NUMBER	INTEGER
SCHEMA_NAME	CHARACTER VARYING (<i>L</i>)
SERVER_NAME	CHARACTER VARYING (<i>L</i>)
Where <i>L</i> is an implementation-defined integer not less than 128 and <i>L1</i> is an implementation-defined integer not less than 254.	

(Continued on next page)

Table 2—Fields in SQL/CLI diagnostics areas (Cont.)

Field	Data type
Fields in status records	
SQLSTATE	CHARACTER (5)
SPECIFIC_NAME	CHARACTER VARYING (<i>L</i>)
SUBCLASS_ORIGIN	CHARACTER VARYING (<i>L1</i>)
TABLE_NAME	CHARACTER VARYING (<i>L</i>)
TRIGGER_CATALOG	CHARACTER VARYING (<i>L</i>)
TRIGGER_NAME	CHARACTER VARYING (<i>L</i>)
TRIGGER_SCHEMA	CHARACTER VARYING (<i>L</i>)
Implementation-defined status field	Implementation-defined
Where <i>L</i> is an implementation-defined integer not less than 128 and <i>L1</i> is an implementation-defined integer not less than 254.	

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

4.3.1 Setting of ROW_NUMBER and COLUMN_NUMBER fields

Unless explicitly set to a different value in this part of ISO/IEC 9075, the ROW_NUMBER and COLUMN_NUMBER fields in a diagnostic record are always 0 (zero).

4.4 Miscellaneous characteristics

4.4.1 Handles

The AllocHandle routine returns a handle, that uniquely identifies the allocated resource. Although the data type for a handle parameter 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.

In general, if the related resource cannot be allocated, then a handle value of zero is returned. However, even if a resource has been successfully allocated, processing of that resource can subsequently fail due to memory constraints as follows:

- If additional memory is required but is not available, then an exception condition is raised: *CLI-specific condition — memory allocation error.*
- If previously allocated memory cannot be accessed, then an exception condition is raised: *CLI-specific condition — memory management error.*

NOTE 2 – No diagnostic information is generated in this case.

The validity of a handle in a compilation unit other than the one in which the identified resource was allocated is implementation-defined.

Specifying (the address of) a valid handle as the output handle of AllocHandle does not have the effect of reinitializing the identified resource. Instead, a new resource is allocated and a new handle value overwrites the old one.

4.4.2 Null terminated strings

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

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

4.4.3 Null pointers

If the standard programming language of the invoking host program supports pointers, then the host program 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 application wishes to prohibit the return of this information.
- In lieu of other output arguments where specifically allowed by this part of ISO/IEC 9075. This indicates that the application wishes to prohibit the return of this information.
- In lieu of input arguments where specifically allowed by this part of ISO/IEC 9075. The semantics of such a specification depend on the context.

If the host program provides a null pointer in any other circumstances, then an exception condition is raised: *CLI-specific condition — invalid use of null pointer*.

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

4.4.4 Environment attributes

Environment attributes are associated with each allocated SQL-environment and affect the behavior of CLI functions in that SQL-environment.

The GetEnvAttr routine enables the application to determine the current value of a specific attribute. For attributes that may be set by the user, the SetEnvAttr routine enables the application to set the value of a specific attribute. Attribute values may be set by the application whenever there are no SQL-connections allocated within the SQL-environment.

Table 15, “Codes used for environment attributes”, and Table 19, “Data types of attributes”, in Subclause 5.14, “Other tables associated with CLI”, indicate for each attribute its name, code value, data type, possible values, and whether the attribute may be set using SetEnvAttr.

The NULL TERMINATION attribute determines whether output character strings are null terminated by the implementation. The attribute is set to true when an SQL-environment is allocated.

4.4.5 Connection attributes

Connection attributes are associated with each allocated SQL-connection and affect the behavior of CLI functions operating in the context of that allocated SQL-connection.

The GetConnectAttr routine enables the application to determine the current value of a specific attribute. For attributes that may be set by the user, the SetConnectAttr routine enables the application to set the value of a specific attribute.

Table 16, “Codes used for connection attributes”, and Table 19, “Data types of attributes”, in Subclause 5.14, “Other tables associated with CLI”, indicate for each attribute its name, code value, data type, possible values and whether the attribute may be set using SetConnectAttr.

The POPULATE IPD attribute determines whether the implementation will populate the implementation parameter descriptor with a descriptor for the <dynamic parameter specification>s when an SQL-statement is prepared or executed immediately. The attribute is automatically set each time an SQL-connection is established for the allocated SQL-connection.

The SAVEPOINT NAME and SAVEPOINT NUMBER connection attributes specify the savepoint to be referenced in an invocation of the EndTran routine that uses the SAVEPOINT NAME COMMIT, SAVEPOINT NAME RELEASE, or SAVEPOINT NUMBER COMMIT, SAVEPOINT NUMBER RELEASE CompletionType, respectively. The SAVEPOINT NAME attribute is set to a zero-length string and the SAVEPOINT NUMBER attribute is set to 0 (zero) when the SQL-connection is allocated.

4.4.6 Statement attributes

Statement attributes are associated with each allocated SQL-statement and affect the processing of SQL-statements under that allocated SQL-statement.

The GetStmtAttr routine enables the application to determine the current value of a specific attribute. For attributes that may be set by the user, the SetStmtAttr routine enables the application to set the value of a specific attribute.

Table 17, “Codes used for statement attributes”, and Table 19, “Data types of attributes”, in Subclause 5.14, “Other tables associated with CLI”, indicate for each attribute its name, code value, data type, possible values, and whether the attribute may be set by using SetStmtAttr.

The APD HANDLE attribute is the value of the handle of the current application parameter descriptor for the allocated SQL-statement. The attribute is set to the value of the handle of the automatically allocated application parameter descriptor when the SQL-statement is allocated.

The ARD HANDLE attribute is the value of the handle of the current application row descriptor for the allocated SQL-statement. The attribute is set to the value of the handle of the automatically allocated application row descriptor when the SQL-statement is allocated.

The IPD HANDLE attribute is the value of the handle of the implementation parameter descriptor associated with the allocated SQL-statement. The attribute is set to the value of the handle of the automatically allocated implementation parameter descriptor when the SQL-statement is allocated.

The IRD HANDLE attribute is the value of the handle of the implementation row descriptor associated with the allocated SQL-statement. The attribute is set to the value of the handle of the automatically allocated implementation row descriptor when the SQL-statement is allocated.

The CURSOR SCROLLABLE attribute determines the *scrollability* implicitly declared when Execute or ExecDirect are invoked. The attribute is set to NONSCROLLABLE when the statement is allocated. The CURSOR SENSITIVITY attribute determines the *sensitivity* to changes of the cursor implicitly declared when Execute or ExecDirect are invoked. The attribute is set to ASENSITIVE when the statement is allocated.

The CURSOR HOLDABLE attribute determines the *holdability* of the cursor implicitly declared when Execute or ExecDirect are invoked. The attribute is set to HOLDABLE or NONHOLDABLE when the statement is allocated, depending on the values of the CURSOR COMMIT BEHAVIOR item used by the GetInfo routine.

The statement attribute CURRENT OF POSITION identifies the row in the rowset to which a positioned update or delete operation applies. This is set to 1 (one) when a statement is initially allocated. It is reset to 1 (one) whenever Fetch or FetchScroll are successfully executed.

The NEST DESCRIPTOR attribute determines whether nested descriptor items are permitted in a CLI descriptor. Nested descriptor items are used to describe ROW and ARRAY data types. The attribute is set to FALSE when the statement is allocated.

4.4.7 CLI descriptor areas

A CLI descriptor area provides an interface for a description of <dynamic parameter specification>s, <dynamic parameter specification> values, resultant columns of a <dynamic select statement> or <dynamic single row select statement>, or the <target specification>s for the resultant columns.

Each descriptor area comprises header fields and zero or more item descriptor areas. The header fields are specified in Table 6, “Fields in SQL/CLI row and parameter descriptor areas”. The header fields include a COUNT fields that indicates the number of item descriptor areas and an ALLOC_TYPE field that indicated whether the CLI descriptor area was allocated by the user or automatically allocated by the implementation.

The header fields include ARRAY_SIZE, ARRAY_STATUS_POINTER, and ROWS_PROCESSED_POINTER. These three fields are used to support the fetching of multiple rows with one invocation of Fetch or FetchScroll.

Each CLI item descriptor area consists of the fields specified in Table 6, “Fields in SQL/CLI row and parameter descriptor areas”, in Subclause 5.13, “Description of CLI item descriptor areas”.

The CLI descriptor areas for the four interface types are referred to as an implementation parameter descriptor (IPD), an application parameter descriptor (APD), an implementation row descriptor (IRD), and an application row descriptor (ARD), respectively.

When an SQL-statement is allocated, a CLI descriptor area of each type is automatically allocated by the implementation. The ALLOC_TYPE fields for these CLI descriptor areas are set to indicate AUTOMATIC. CLI descriptor areas allocated by the user have their ALLOC_TYPE fields set to indicate USER, and can only be used as an APD or ARD. The handle values of the IPD, IRD, current APD, and current ARD are attributes of the allocated SQL-statement. The application can determine the current values of these attributes by using the routine GetStmtAttr. The current APD and ARD are initially the automatically-allocated APD and ARD, respectively, but can subsequently be changed by changing the corresponding attribute value using the routine SetStmtAttr.

The routines GetDescField and GetDescRec enable information to be retrieved from any CLI descriptor area. The routines SetDescField and SetDescRec enable information to be set in any CLI descriptor area except an IRD. The routine BindCol implicitly sets information in the current ARD. The routine BindParameter implicitly sets information in the current APD and the current IPD. The CopyDesc routine enables the contents of any CLI descriptor area to be copied to any CLI descriptor area except an IRD.

4.4 Miscellaneous characteristics

NOTE 3 – Although there is no need to set a DATA_POINTER field in the IPD, to align with the consistency check that applies in the case of an APD or ARD, setting this field causes the item descriptor area to be validated.

4.4.8 Obtaining diagnostics during multi-row fetch

When Fetch or FetchScroll is used to fetch a rowset, exceptions or warnings may be raised during the retrieval of one or more rows in the rowset. The status of each row (that is, information about whether each row in the rowset was successfully retrieved or not) is available in the array addressed by the ARRAY_STATUS_POINTER field of the applicable IRD. The cardinality of this array is the same as the ARRAY_SIZE field of the corresponding ARD; for each row in the rowset, the corresponding element of this array has one of the following values:

- A value of 0 (zero), indicates **Row success**, meaning that the row was fetched successfully.
- A value of 6, indicates **Row success with information**, meaning that the row was fetched successfully, but a completion condition was raised: *warning*.
- A value of 3, indicates **No row**, meaning that there is no row at this position in the rowset. This condition occurs when a partial rowset is retrieved because the result set ended.
- A value of 5, indicates **Row error**, meaning that the row was not fetched successfully and an exception condition was raised.

Each **Row success with information** or **Row Error** generates one or more status records in the diagnostics area. The ROW_NUMBER field for each status record has the value of the row position within the rowset to which this status record corresponds.

4.5 Client-server operation

New paragraph If the execution of a CLI routine causes the implicit or explicit execution of an <SQL procedure statement> by an SQL-server, diagnostic information is passed in an implementation-dependent manner to the SQL-client and then into the appropriate diagnostics area. The effect on diagnostic information of incompatibilities between the character repertoires supported by the SQL-client and the SQL-server is implementation-dependent.

5 Call-Level Interface specifications

5.1 <CLI routine>

Function

Describe SQL/CLI routines in a generic fashion.

Format

```

<CLI routine> ::=
    <CLI routine name>
    <CLI parameter list>
    [ <CLI returns clause> ]

<CLI routine name> ::= <CLI name prefix><CLI generic name>

<CLI name prefix> ::=
    <CLI by-reference prefix>
    | <CLI by-value prefix>

<CLI by-reference prefix> ::= SQLR

<CLI by-value prefix> ::= SQL

<CLI generic name> ::=
    AllocConnect
    | AllocEnv
    | AllocHandle
    | AllocStmt
    | BindCol
    | BindParameter
    | Cancel
    | CloseCursor
    | ColAttribute
    | ColumnPrivileges
    | Columns
    | Connect
    | CopyDesc
    | DataSources
    | DescribeCol
    | Disconnect
    | EndTran
    | Error
    | ExecDirect
    | Execute
    | Fetch
    | FetchScroll
    | ForeignKeys
    | FreeConnect
    | FreeEnv
    | FreeHandle
    | FreeStmt
    | GetConnectAttr
    | GetCursorName

```

5.1 <CLI routine>

```

| GetData
| GetDescField
| GetDescRec
| GetDiagField
| GetDiagRec
| GetEnvAttr
| GetFeatureInfo
| GetFunctions
| GetInfo
| GetLength
| GetParamData
| GetPosition
| GetSessionInfo
| GetStmtAttr
| GetSubString
| GetTypeInfo
| MoreResults
| NextResult
| NumResultCols
| ParamData
| Prepare
| PrimaryKeys
| PutData
| RowCount
| SetConnectAttr
| SetCursorName
| SetDescField
| SetDescRec
| SetEnvAttr
| SetStmtAttr
| SpecialColumns
| StartTran
| TablePrivileges
| Tables
| <implementation-defined CLI generic name>

```

```

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

```

```

<CLI parameter declaration> ::=
    <CLI parameter name> <CLI parameter mode> <CLI parameter data type>

```

<CLI parameter name> ::= !! *See the individual CLI routine definitions*

```

<CLI parameter mode> ::=
    IN
    | OUT
    | DEFIN
    | DEFOUT
    | DEF

```

```

<CLI parameter data type> ::=
    INTEGER
    | SMALLINT
    | ANY
    | CHARACTER <left paren> <length> <right paren>

```

<CLI returns clause> ::= RETURNS SMALLINT

<implementation-defined CLI generic name> ::= !! *See the Syntax Rules*

Syntax Rules

- 1) <CLI routine> is a pre-defined routine written in a standard programming language that is invoked by a compilation unit of the same standard programming language. Let *HL* be that standard programming language. *HL* shall be one of Ada, C, COBOL, Fortran, MUMPS, Pascal, or PL/I.
- 2) <CLI routine> that contains a <CLI returns clause> is called a *CLI function*. A <CLI routine> that does not contain a <CLI returns clause> is called a *CLI procedure*.
- 3) For each CLI function *CF*, there is a corresponding CLI procedure *CP*, with the same <CLI routine name>. The <CLI parameter list> for *CP* is the same as the <CLI parameter list> for *CF* but with the following additional <CLI parameter declaration>:

ReturnCode OUT SMALLINT

- 4) *HL* shall support either the invocation of *CF* or the invocation of *CP*. It is implementation-defined which is supported.
- 5) Case:
 - a) If <CLI parameter mode> is IN, then the parameter is an *input parameter*. The value of an input argument is established when a CLI routine is invoked.
 - b) If <CLI parameter mode> is OUT, then the parameter is an *output parameter*. The value of an output argument is established when a CLI routine is executed.
 - c) If <CLI parameter mode> is DEFIN, then the parameter is a *deferred input parameter*. The value of a deferred input argument for a CLI routine *R* is not established when *R* is invoked, but subsequently during the execution of a related CLI routine.
 - d) If <CLI parameter mode> is DEFOUT, then the parameter is a *deferred output parameter*. The value of a deferred output argument for a CLI routine *R* is not established by the execution of *R* but subsequently by the execution of a related CLI routine.
 - e) If <CLI parameter mode> is DEF, then the parameter is a *deferred parameter*. The value of a deferred argument for a CLI routine *R* is not established by the execution of *R* but subsequently by the execution of a related CLI routine.
- 6) The value of an output, deferred output, deferred input, or deferred parameter is an address. It is either a non-pointer host variable passed by reference or a pointer host variable passed by value.
- 7) A *by-value version* of a CLI routine is a version that expects each of its non-character input parameters to be provided as actual values. A *by-reference version* of a CLI routine is a version that expects each of its input parameters to be provided as an address. By-value and by-reference versions of the CLI routines shall be supported according to Table 3, "Supported calling conventions of SQL/CLI routines by language".

Table 3—Supported calling conventions of SQL/CLI routines by language

Language	By-value	By-reference
Ada (ISO 8652)	Optional	Required
C (ISO/IEC 9899)	Required	Optional
COBOL (ISO 1989)	Optional	Required
Fortran (ISO/IEC 1539)	Not supported	Required
MUMPS (ISO/IEC 11756)	Optional	Required
Pascal (ISO/IEC 7185 and ISO/IEC 10206)	Optional	Required
PL/I (ISO 6160)	Optional	Required

- 8) If a <CLI routine> is a by-reference routine, then its <CLI routine name> shall contain a <CLI by-reference prefix>. Otherwise, its <CLI routine name> shall contain a <CLI by-value prefix>.
- 9) The <implementation-defined CLI generic name> for an implementation-defined CLI function shall be different from the <CLI generic name> of any other CLI function. The <implementation-defined CLI generic name> for an implementation-defined CLI procedure shall be different from the <CLI generic name> of any other CLI procedure.
- 10) Any <CLI routine name> that cannot be used by an implementation because of its length or because it is made identical to some other <CLI routine name> by truncation is effectively replaced with an abbreviated name according to the following rules:
 - a) Any <CLI by-value prefix> remains unchanged.
 - b) Any <CLI by-reference prefix> is replaced by SQR.
 - c) The <CLI generic name> is replaced by an abbreviated version according to Table 4, “Abbreviated SQL/CLI generic names”.

Table 4—Abbreviated SQL/CLI generic names

Generic Name	Abbreviation
AllocConnect	AC
AllocEnv	AE
AllocHandle	AH
AllocStmt	AS
BindCol	BC
BindParameter	BP
Cancel	CAN
CloseCursor	CC
ColAttribute	CO
ColumnPrivileges	CP

Table 4—Abbreviated SQL/CLI generic names (Cont.)

Generic Name	Abbreviation
Columns	COL
Connect	CON
CopyDesc	CD
DataSources	DS
DescribeCol	DC
Disconnect	DIS
EndTran	ET
Error	ER
ExecDirect	ED
Execute	EX
Fetch	FT
FetchScroll	FTS
ForeignKeys	FK
FreeConnect	FC
FreeEnv	FE
FreeHandle	FH
FreeStmt	FS
GetConnectAttr	GCA
GetCursorName	GCN
GetData	GDA
GetDescField	GDF
GetDescRec	GDR
GetDiagField	GXF
GetDiagRec	GXR
GetEnvAttr	GEA
GetFeatureInfo	GFI
GetFunctions	GFU
GetInfo	GI
GetLength	GLN
GetParamData	GPD
GetPosition	GPO
GetSessionInfo	GSI
GetStmtAttr	GSA
GetSubString	GSB

5.1 <CLI routine>

Table 4—Abbreviated SQL/CLI generic names (Cont.)

Generic Name	Abbreviation
GetTypeInfo	GTI
MoreResults	MR
NextResult	NR
NumResultCols	NRC
ParamData	PRD
Prepare	PR
PrimaryKeys	PK
PutData	PTD
RowCount	RC
SetConnectAttr	SCA
SetCursorName	SCN
SetDescField	SDF
SetDescRec	SDR
SetEnvAttr	SEA
SetStmtAttr	SSA
SpecialColumns	SC
StartTran	STN
TablePrivileges	TP
Tables	TAB
Implementation-defined CLI routine	Implementation-defined abbreviation

- 11) Let *CR* be a <CLI routine> and let *RN* be its <CLI 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 *CR* shall be *RN*.
 - b) If *HL* does not support <simple Latin lower case letter>s, then the name used for the invocation of *CR* shall be *RNU*.
 - c) If *HL* does not support case sensitive routine names, then the name used for the invocation of *CR* shall be *RN* or *RNU*.
- 12) Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 5.15, “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”.

- 13) 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 CLI parameter data type is INTEGER, then the type of the corresponding argument shall be TI .
 - ii) If the CLI parameter data type is SMALLINT, then the type of the corresponding argument shall be TS .
 - iii) If the CLI parameter data type is CHARACTER(L), then the type of the corresponding argument shall be TC .
 - iv) If the CLI 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 CLI routine is a CLI function, then the type of the returned value is TS .

Access Rules

None.

General Rules

- 1) The rules for invocation of a <CLI routine> are specified in Subclause 5.2, “<CLI routine> invocation”.

5.2 <CLI routine> invocation

Function

Specify the rules for invocation of a <CLI routine>.

Syntax Rules

- 1) Let *HL* be the standard programming language of the invoking host program.
- 2) A CLI function or CLI procedure is invoked by the *HL* mechanism for invoking functions or procedures, respectively.
- 3) Let *RN* be the <CLI routine name> of the <CLI routine> invoked by the host program. The number of arguments provided in the invocation shall be the same as the number of <CLI 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 <CLI parameter data type> of the *i*-th <CLI parameter declaration> of *RN*. *DA* shall be the *HL* equivalent of *DP* as specified by the rules of Subclause 5.1, "<CLI routine>".

General Rules

- 1) If the value of any input argument provided by the host program 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 <CLI routine> falls outside the set of values supported by the host program for that parameter, then the effect is implementation-defined.
- 2) Let *GRN* be the <CLI generic name> of *RN*.
- 3) When the <CLI routine> is called by the host program:
 - a) The values of all input arguments to *RN* are established.
 - b) Case:
 - i) If *RN* is a CLI routine with a statement handle as an input parameter, *RN* has no accompanying handle type parameter, and *GRN* is not 'Error', then:
 - 1) If the statement handle does not identify an allocated SQL-statement, then an exception condition is raised: *CLI-specific condition — invalid handle*. Otherwise, let *S* be the allocated SQL-statement identified by the statement handle.
 - 2) If *GRN* is not 'Cancel', then the diagnostics area associated with *S* is emptied.
 - 3) Let *C* be the allocated SQL-connection with which *S* is associated.
 - 4) If there is no established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection does not exist*. Otherwise, let *EC* be the established SQL-connection associated with *C*.
 - 5) If *EC* is not the current connection, then the General Rules of Subclause 5.3, "Implicit set connection", are applied to *EC* as the dormant connection.

- 6) If *GRN* is neither 'Cancel' nor 'ParamData' nor 'PutData' and there is a deferred parameter number associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.
- 7) *RN* is invoked.
- ii) If *RN* is a CLI routine with a descriptor handle as an input parameter and *RN* has no accompanying handle type parameter and *GRN* is not 'CopyDesc', then:
 - 1) If the descriptor handle does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*. Otherwise, let *D* be the allocated CLI descriptor area identified by the descriptor handle.
 - 2) The diagnostics area associated with *D* is emptied.
 - 3) Let *C* be the allocated SQL-connection with which *D* is associated.
 - 4) If there is no established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection does not exist*. Otherwise, let *EC* be the established SQL-connection associated with *C*.
 - 5) If *EC* is not the current connection, then the General Rules of Subclause 5.3, "Implicit set connection", are applied to *EC* as the dormant connection.
 - 6) *RN* is invoked.
- iii) Otherwise, *RN* is invoked.
- 4) Case:
 - a) If the <CLI routine> is a CLI function, then:
 - i) The values of all output arguments are established.
 - ii) Let *RC* be the return value.
 - b) If the <CLI routine> is a CLI 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.
- 5) Case:
 - a) If *RN* did not complete execution because it requires more input data, then:
 - i) *RC* is set to indicate **Data needed**.
 - ii) An exception condition is raised: *CLI-specific condition — dynamic parameter value needed*.
 - b) If *RN* executed successfully, then:
 - i) Either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*.

- ii) Case:
 - 1) If a completion condition is raised: *successful completion*, then *RC* is set to indicate **Success**.
 - 2) If a completion condition is raised: *warning*, then *RC* is set to indicate **Success with information**.
 - 3) If a completion condition is raised: *no data*, then *RC* is set to indicate **No data found**.
- c) If *RN* did not execute successfully, then:
 - i) All changes made to SQL-data or schemas by the execution of *RN* are canceled.
 - ii) One or more exception conditions are raised as determined by the General Rules of this and other Subclauses of this part of ISO/IEC 9075 or by implementation-defined rules.
- iii) Case:
 - 1) If an exception condition is raised: *CLI-specific condition — invalid handle*, then *RC* is set to indicate **Invalid handle**.
 - 2) Otherwise, *RC* is set to indicate **Error**.
- 6) Case:
 - a) If *GRN* is neither 'Error', 'GetDiagField', nor 'GetDiagRec' and *RC* indicates neither **Invalid handle** nor **Data needed**, then diagnostic information resulting from the execution of *RN* is placed into the appropriate diagnostics area as specified in Subclause 4.2, "Return codes", and Subclause 4.3, "Diagnostics areas in SQL/CLI".
 - b) Otherwise, no diagnostics area is updated.