

Australian Standard™

**Information technology—Database
languages—SQL**

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



This Australian Standard was prepared by Committee IT-027, Data Management and Interchange. It was approved on behalf of the Council of Standards Australia on 16 May 2005. This Standard was published on 16 June 2005.

The following are represented on Committee IT-027:

Association of Superannuation Funds of Australia
Australia Post
Australia Bankers Association
Australian Bureau of Statistics
Australian Computer Society
Australian Customs Service
Australian Electoral Commission
Australian Electric and Electronic Manufacturers Association
Australian Industry Group
Australian Information Industry Association
Australian Institute of Health and Welfare
Australian Taxation Office
Centrelink
Department of Defence
Department of Immigration, Multicultural and Indigenous Affairs
Health Insurance Commission
Data Management Association Australia

Keeping Standards up-to-date

Standards are living documents which reflect progress in science, technology and systems. To maintain their currency, all Standards are periodically reviewed, and new editions are published. Between editions, amendments may be issued. Standards may also be withdrawn. It is important that readers assure themselves they are using a current Standard, which should include any amendments which may have been published since the Standard was purchased.

Detailed information about Standards can be found by visiting the Standards Web Shop at www.standards.com.au and looking up the relevant Standard in the on-line catalogue.

Alternatively, the printed Catalogue provides information current at 1 January each year, and the monthly magazine, *The Global Standard*, has a full listing of revisions and amendments published each month.

Australian Standards™ and other products and services developed by Standards Australia are published and distributed under contract by SAI Global, which operates the Standards Web Shop.

We also welcome suggestions for improvement in our Standards, and especially encourage readers to notify us immediately of any apparent inaccuracies or ambiguities. Contact us via email at mail@standards.org.au, or write to the Chief Executive, Standards Australia, GPO Box 5420, Sydney, NSW 2001.

This Standard was issued in draft form for comment as DR 05144.

Australian Standard™

Information technology—Database languages—SQL

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

Originated as AS/NZS 3968.3:1998.
Revised and redesignated as AS ISO/IEC 9075.3—2005.

COPYRIGHT

© Standards Australia

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher.

Published by Standards Australia GPO Box 5420, Sydney, NSW 2001, Australia

ISBN 0 7337 6765 6

PREFACE

This Standard was prepared by the Standards Australia Committee IT-027, Data Management and Interchange to supersede AS/NZS 3968.3:1998.

This Standard is identical with, and has been reproduced from ISO/IEC 9075-3:2003, *Information technology—Database languages—SQL—Part 3: Call-Level Interface (SQL/CLI)*.

The objective of this Standard is to provide database designers, administrators and developers with the definition of structures and procedures 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.

This Standard is Part 3 of AS 9075, *Information technology—Database languages—SQL*, which is published in parts as follows:

- Part 1: Framework (SQL/Framework)
- Part 2: Foundation (SQL/Foundation)
- Part 3: Call-Level Interface (SQL/CLI) (this Standard)
- Part 4: Persistent Stored Modules (SQL/PSM)
- Part 5: Host Language Bindings (SQL/Bindings)
- Part 9: Management of External Data (SQL/MED)
- Part 10: Object Language Bindings (SQL/OLB)
- Part 11: Information and Definition Schemas (SQL/Schemata)
- Part 13: SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)
- Part 14: XML-Related Specifications (SQL/XML)

As this Standard is reproduced from an international standard, the following applies:

- (a) Its number appears on the cover and title page while the international standard number appears only on the cover.
- (b) In the source text ‘this part of ISO/IEC 9075’ should read ‘this Australian Standard’.
- (c) A full point substitutes for a comma when referring to a decimal marker.

References to International Standards should be replaced by references to Australian or Australian/New Zealand Standards, as follows:

<i>Reference to International Standard</i>		<i>Australian Standard</i>	
ISO/IEC		AS	
7185	Information technology—Programming languages—Pascal	2580	Programming languages—Pascal
9075	Information technology—Database languages—SQL	9075	Information technology—Database languages—SQL
9075-1	Part 1: Framework (SQL/Framework)	9075.1	Part 1: Framework (SQL/Framework)
9075-2	Part 2: Foundation (SQL/Foundation)	9075.2	Part 2: Foundation (SQL/Foundation)
9075-11	Part 11: Information and Definition Schemas (SQL/Schemas)	9075.11	Part 11: Information and Definition Schemas (SQL/Schemas)
10206	Information technology—Programming languages—Extended pascal	3981	Information technology—Programming languages—Extended pascal

Only referenced documents that have been adopted as Australian or Australian/New Zealand Standards have been listed.

CONTENTS

Page

1	Scope.....	1
2	Normative references.....	3
2.1	JTC1 standards.....	3
3	Definitions, notations, and conventions.....	5
3.1	Definitions.....	5
3.1.1	Definitions provided in Part 3.....	5
3.2	Conventions.....	5
3.2.1	Specification of routine definitions.....	5
4	Concepts.....	7
4.1	Introduction to SQL/CLI.....	7
4.2	Return codes.....	10
4.3	Diagnostics areas in SQL/CLI.....	11
4.3.1	Setting of ROW_NUMBER and COLUMN_NUMBER fields.....	14
4.4	Miscellaneous characteristics.....	15
4.4.1	Handles.....	15
4.4.2	Null terminated strings.....	15
4.4.3	Null pointers.....	15
4.4.4	Environment attributes.....	16
4.4.5	Connection attributes.....	16
4.4.6	Statement attributes.....	17
4.4.7	CLI descriptor areas.....	18
4.4.8	Obtaining diagnostics during multi-row fetch.....	19
4.5	Client-server operation.....	19
5	Call-Level Interface specifications.....	21
5.1	<CLI routine>.....	21
5.2	<CLI routine> invocation.....	30
5.3	Implicit set connection.....	33
5.4	Implicit cursor.....	34
5.5	Implicit DESCRIBE USING clause.....	36
5.6	Implicit EXECUTE USING and OPEN USING clauses.....	42
5.7	Implicit CALL USING clause.....	48
5.8	Implicit FETCH USING clause.....	52
5.9	Character string retrieval.....	58
5.10	Binary large object string retrieval.....	59

5.11	Deferred parameter check.	60
5.12	CLI-specific status codes.	61
5.13	Description of CLI item descriptor areas.	64
5.14	Other tables associated with CLI.	76
5.15	SQL/CLI data type correspondences.	105
6	SQL/CLI routines.	115
6.1	AllocConnect.	115
6.2	AllocEnv.	116
6.3	AllocHandle.	117
6.4	AllocStmt.	121
6.5	BindCol.	122
6.6	BindParameter.	124
6.7	Cancel.	129
6.8	CloseCursor.	131
6.9	ColAttribute.	132
6.10	ColumnPrivileges.	134
6.11	Columns.	140
6.12	Connect.	150
6.13	CopyDesc.	154
6.14	DataSources.	155
6.15	DescribeCol.	157
6.16	Disconnect.	159
6.17	EndTran.	161
6.18	Error.	165
6.19	ExecDirect.	167
6.20	Execute.	170
6.21	Fetch.	172
6.22	FetchScroll.	175
6.23	ForeignKeys.	179
6.24	FreeConnect.	192
6.25	FreeEnv.	193
6.26	FreeHandle.	194
6.27	FreeStmt.	197
6.28	GetConnectAttr.	199
6.29	GetCursorName.	201
6.30	GetData.	202
6.31	GetDescField.	209
6.32	GetDescRec.	211
6.33	GetDiagField.	213
6.34	GetDiagRec.	222
6.35	GetEnvAttr.	225
6.36	GetFeatureInfo.	227

6.37	GetFunctions.	230
6.38	GetInfo.	231
6.39	GetLength.	239
6.40	GetParamData.	241
6.41	GetPosition.	247
6.42	GetSessionInfo.	250
6.43	GetStmtAttr.	252
6.44	GetSubString.	255
6.45	GetTypeInfo.	258
6.46	MoreResults.	262
6.47	NextResult.	264
6.48	NumResultCols.	265
6.49	ParamData.	266
6.50	Prepare.	273
6.51	PrimaryKeys.	275
6.52	PutData.	280
6.53	RowCount.	283
6.54	SetConnectAttr.	284
6.55	SetCursorName.	286
6.56	SetDescField.	288
6.57	SetDescRec.	293
6.58	SetEnvAttr.	295
6.59	SetStmtAttr.	297
6.60	SpecialColumns.	301
6.61	StartTran.	308
6.62	TablePrivileges.	310
6.63	Tables.	315
7	Definition Schema.	323
7.1	SQL_IMPLEMENTATION_INFO base table.	323
7.2	SQL_SIZING base table.	326
7.3	SQL_LANGUAGES base table.	328
8	Conformance.	329
8.1	Claims of conformance to SQL/CLI.	329
8.2	Additional conformance requirements for SQL/CLI.	329
8.3	Implied feature relationships of SQL/CLI.	330
Annex A	Typical header files.	331
A.1	C header file SQLCLI.H.	331
A.2	COBOL library item SQLCLI.	344
Annex B	Sample C programs.	355
B.1	Create table, insert, select.	355
B.2	Interactive Query.	358

B.3	Providing long dynamic arguments at Execute time.	362
Annex C	Implementation-defined elements.	365
Annex D	Implementation-dependent elements.	379
Annex E	Incompatibilities with ISO/IEC 9075:1999.	385
Annex F	SQL feature taxonomy.	387
Index.		389

TABLES

Page

1	Fields in SQL/CLI diagnostics areas.	12
2	Supported calling conventions of SQL/CLI routines by language.	24
3	Abbreviated SQL/CLI generic names.	24
4	SQLSTATE class and subclass values for SQL/CLI-specific conditions.	61
5	Fields in SQL/CLI row and parameter descriptor areas.	69
6	Codes used for implementation data types in SQL/CLI.	72
7	Codes used for application data types in SQL/CLI.	73
8	Codes associated with datetime data types in SQL/CLI.	74
9	Codes associated with <interval qualifier> in SQL/CLI.	74
10	Codes associated with <parameter mode> in SQL/CLI.	75
11	Codes associated with user-defined types in SQL/CLI.	75
12	Codes used for SQL/CLI diagnostic fields.	76
13	Codes used for SQL/CLI handle types.	78
14	Codes used for transaction termination.	78
15	Codes used for environment attributes.	78
16	Codes used for connection attributes.	79
17	Codes used for statement attributes.	79
18	Codes used for FreeStmt options.	79
19	Data types of attributes.	80
20	Codes used for SQL/CLI descriptor fields.	80
21	Ability to set SQL/CLI descriptor fields.	83
22	Ability to retrieve SQL/CLI descriptor fields.	86
23	SQL/CLI descriptor field default values.	88
24	Codes used for fetch orientation.	90
25	Multi-row fetch status codes.	91
26	Miscellaneous codes used in CLI.	91
27	Codes used to identify SQL/CLI routines.	92
28	Codes and data types for implementation information.	95
29	Codes and data types for session implementation information.	97
30	Values for ALTER TABLE with GetInfo.	98
31	Values for FETCH DIRECTION with GetInfo.	98
32	Values for GETDATA EXTENSIONS with GetInfo.	98
33	Values for OUTER JOIN CAPABILITIES with GetInfo.	98
34	Values for SCROLL CONCURRENCY with GetInfo.	99
35	Values for TRANSACTION ISOLATION OPTION with GetInfo and StartTran.	99
36	Values for TRANSACTION ACCESS MODE with StartTran.	99
37	Codes used for concise data types.	100
38	Codes used with concise datetime data types in SQL/CLI.	101
39	Codes used with concise interval data types in SQL/CLI.	102

40	Concise codes used with datetime data types in SQL/CLI.....	102
41	Concise codes used with interval data types in SQL/CLI.....	103
42	Special parameter values.....	103
43	Column types and scopes used with SpecialColumns.....	104
44	SQL/CLI data type correspondences for Ada.....	105
45	SQL/CLI data type correspondences for C.....	106
46	SQL/CLI data type correspondences for COBOL.....	107
47	SQL/CLI data type correspondences for Fortran.....	109
48	SQL/CLI data type correspondences for M.....	110
49	SQL/CLI data type correspondences for Pascal.....	112
50	SQL/CLI data type correspondences for PL/I.....	113
51	Implied feature relationships of SQL/CLI.....	330
52	Feature taxonomy and definition for mandatory features.....	387

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 in 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 9075:1999”](#), is an informative Annex. It identifies incompatibilities with ISO/IEC 9075-3:1995.
- 14) [Annex F, “SQL feature 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.

X

NOTES

AUSTRALIAN STANDARD

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.

This page intentionally left blank.

2 Normative references

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

2.1 JTC1 standards

[ISO1539] ISO/IEC 1539-1:1997, *Information technology — Programming languages — Fortran — Part 1: Base language*.

ISO/IEC 1539-1:1997/Cor.1:2001.

ISO/IEC 1539-1:1997/Cor.2:2002.

[ISO1989] ISO 1989:1985, *Programming languages — COBOL*. (Endorsement of ANSI X3.23-1985).

ISO/IEC 1989:1985/Amd.1:1992, *Intrinsic function module*

ISO/IEC 1989:1985/Amd.2:1994, *Correction and clarification amendment for COBOL*

[ISO6160] ISO 6160:1979, *Programming languages — PL/I*. (Endorsement of ANSI X3.53-1976).

[ISO7185] ISO/IEC 7185:1990, *Information technology — Programming languages — Pascal*.

[ISO8652] ISO/IEC 8652:1995, *Information technology — Programming languages — Ada*.

ISO/IEC 8652:1995/Cor.1:2001.

[Framework] ISO/IEC 9075-1:2003, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

[Foundation] ISO/IEC 9075-2:2003, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

[Schemata] ISO/IEC 9075-11:2003, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*.

[ISO9899] ISO/IEC 9899:1999, *Programming languages — C*.

ISO/IEC 9899:1999/Cor 1:2001, *Technical Corrigendum to ISO/IEC 9899:1999*.

[ISO10206] ISO/IEC 10206:1991, *Information technology — Programming languages — Extended Pascal*.

[ISO11756] ISO/IEC 11756:1999, *Information technology — Programming languages — M*.

This page intentionally left blank.

3 Definitions, notations, and conventions

This Clause modifies Clause 3, “Definitions, notations, and conventions”, in ISO/IEC 9075-2.

3.1 Definitions

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

3.1.1 Definitions provided in Part 3

For the purposes of this part of ISO/IEC 9075, the following definitions apply.

- 3.1.1.1 data source:** A synonym for the SQL-server that is part of the current SQL-connection.
- 3.1.1.2 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.
- 3.1.1.3 inner table:** The second operand of a left outer join or the first operand of a right outer join.
- 3.1.1.4 pseudo-column:** A column that is part of a table but is not part of the descriptor for that table. An example of such a pseudo-column is an implementation-defined row identifier.
- 3.1.1.5 SQL/CLI application:** An application that invokes <CLI routine>s specified in this part of ISO/IEC 9075.

3.2 Conventions

This Subclause modifies Subclause 3.3, “Conventions”, in ISO/IEC 9075-2.

3.2.1 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 name, mode, and data type of each 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.

4 Concepts

This Clause modifies [Clause 4](#), “*Concepts*”, in ISO/IEC 9075-2.

4.1 Introduction to SQL/CLI

The Call-Level Interface (SQL/CLI) is a binding style for executing SQL statements. This part of ISO/IEC 9075 provides specifications for 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 SQL/CLI implementation and the SQL-implementation.

A *handle* is a CLI object returned by an SQL/CLI implementation when a CLI resource is allocated; the handle is 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; when invoked, it 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. CLI descriptor areas and SQL-statements 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 SQL/CLI implementation. The SQL/CLI 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, which becomes the *current SQL-connection*. The `Disconnect` routine terminates an established SQL-connection. Switching between established SQL-connections occurs automatically whenever the SQL/CLI application switches processing to a dormant SQL-connection, which then becomes the *current 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 all three cases, the executed SQL-statement can contain dynamic parameters.

The interface for a description of dynamic parameters, dynamic parameter values, the result columns of a <dynamic select statement> or <dynamic single row select statement>, and the target specifications for the

result 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 SQL/CLI 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 SQL/CLI 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 result columns is automatically provided in the applicable CLI implementation descriptor area. In this case, the SQL/CLI application may additionally retrieve information by using the `DescribeCol` and/or the `ColAttribute` routine to obtain a description of a single result column and by using the `NumResultCols` routine to obtain a count of the number of result columns. The SQL/CLI application sets values in the CLI application descriptor area for the description of the corresponding target specifications either explicitly, by using the routines `SetDescField` and `SetDescRec`, or implicitly, by 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 shall be defined in the application descriptor area before SQL-statement execution. This can be done either explicitly, by using the routines `SetDescField` and `SetDescRec`, or implicitly, by 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 piece by piece.

Before a <call statement> is prepared or executed immediately, the SQL/CLI application may choose whether or not to bind any dynamic output parameters in the CLI application descriptor area. This can be done either explicitly, by using the routines `SetDescField` and `SetDescRec`, or implicitly, by 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 SQL/CLI application by using the routine `SetCursorName`. If a cursor name is not supplied by the SQL/CLI 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. At the time that the cursor is implicitly declared, the value of the `CURSOR SCROLLABLE` statement attribute shall be `SCROLLABLE`, allowing the use of `FetchScroll` with a `FetchOrientation` other than `NEXT`. The SQL/CLI 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 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 1 (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 SQL/CLI application as a result of invoking the Execute or ExecDirect routine, supplying 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 Schema. 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 whose names 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 are referenced by 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 whose names 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 whose names 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) can be obtained by executing appropriate queries on the views of the Information Schema.

The GetPosition, GetLength, and GetSubString routines can each be used with its 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 to determine whether a given substring exists within that string and, if it does, to obtain an integer value that indicates the starting position of the first appearance of the given substring.
- The GetLength routine may be used to obtain the length of that string as an integer.
- 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.

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 — Applications are prohibited from using the ExecDirect or Execute routines to execute <start transaction statement>s, <commit statement>s, <rollback statement>s, and <release savepoint statement>s.

The Cancel routine is used to cancel the execution of a concurrently executing SQL/CLI routine; it is also used 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 SQL/CLI 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 invoking 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 return code values and meanings are described in the following list. 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 every output argument that corresponds to an output parameter whose value is not explicitly defined by this part of ISO/IEC 9075 is implementation-dependent.

In addition to providing the return code, for all CLI routines other than Error, GetDiagField, and GetDiagRec, the SQL/CLI 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 takes two input handles, the resource being utilized is the one identified by TargetDescHandle.

4.3 Diagnostics areas in SQL/CLI

Each diagnostics area comprises header information consisting of fields that contain general information relating to the routine that was executed and zero (0) 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.

When Fetch or FetchScroll is invoked, the returned result set has one or more rows, and exceptions or warnings generated, then the corresponding records in the diagnostics area have the ROW_NUMBER field set to the row number of the row in the rowset associated with the exceptions or warnings. 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 status record contains the column number (if any) to which this exception or warning condition applies. If the status 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 the second and subsequent of those status records appear is implementation-dependent. Which of those status records appears first is also implementation-dependent, except that:

- 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 SQL/CLI 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 status records to be generated.

A CLI diagnostics area comprises the header fields specified under “Header fields” Table 1, “Fields in SQL/CLI diagnostics areas”, as well as zero (0) or more status records, each of which comprises the fields specified under “Status record fields” in that same table.

Table 1 — Fields in SQL/CLI diagnostics areas

Field	Data type
Header fields	
DYNAMIC_FUNCTION	CHARACTER VARYING (L1) [†]
DYNAMIC_FUNCTION_CODE	INTEGER
MORE	INTEGER
NUMBER	INTEGER

Field	Data type
RETURNCODE	SMALLINT
ROW_COUNT	INTEGER
TRANSACTIONS_COMMITTED	INTEGER
TRANSACTIONS_ROLLED_BACK	INTEGER
TRANSACTION_ACTIVE	INTEGER
Implementation-defined header field	Implementation-defined data type
Status record fields	
CATALOG_NAME	CHARACTER VARYING (L) [†]
CLASS_ORIGIN	CHARACTER VARYING (L1) [†]
COLUMN_NAME	CHARACTER VARYING (L) [†]
COLUMN_NUMBER	INTEGER
CONDITION_IDENTIFIER	CHARACTER VARYING (L) [†]
CONDITION_NUMBER	INTEGER
CONNECTION_NAME	CHARACTER VARYING (L) [†]
CONSTRAINT_CATALOG	CHARACTER VARYING (L) [†]
CONSTRAINT_NAME	CHARACTER VARYING (L) [†]
CONSTRAINT_SCHEMA	CHARACTER VARYING (L) [†]
CURSOR_NAME	CHARACTER VARYING (L) [†]
MESSAGE_LENGTH	INTEGER
MESSAGE_OCTET_LENGTH	INTEGER
MESSAGE_TEXT	CHARACTER VARYING (L1) [†]
NATIVE_CODE	INTEGER
PARAMETER_MODE	CHARACTER VARYING (L) [†]

Field	Data type
PARAMETER_NAME	CHARACTER VARYING (L) [†]
PARAMETER_ORDINAL_POSITION	INTEGER
ROUTINE_CATALOG	CHARACTER VARYING (L) [†]
ROUTINE_NAME	CHARACTER VARYING (L) [†]
ROUTINE_SCHEMA	CHARACTER VARYING (L) [†]
ROW_NUMBER	INTEGER
SCHEMA_NAME	CHARACTER VARYING (L) [†]
SERVER_NAME	CHARACTER VARYING (L) [†]
SQLSTATE	CHARACTER (5)
SPECIFIC_NAME	CHARACTER VARYING (L) [†]
SUBCLASS_ORIGIN	CHARACTER VARYING (L1) [†]
TABLE_NAME	CHARACTER VARYING (L) [†]
TRIGGER_CATALOG	CHARACTER VARYING (L) [†]
TRIGGER_NAME	CHARACTER VARYING (L) [†]
TRIGGER_SCHEMA	CHARACTER VARYING (L) [†]
Implementation-defined status field	Implementation-defined data type
[†] 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

Except where otherwise specified in this part of ISO/IEC 9075, the ROW_NUMBER and COLUMN_NUMBER fields in a status 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 of 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 for an invocation 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 SQL/CLI 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 SQL/CLI 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 SQL/CLI application supports pointers, then the SQL/CLI application 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 SQL/CLI 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 SQL/CLI 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 SQL/CLI application 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 SQL/CLI application to determine the current value of a specific attribute. For attributes that may be set by the user, the SetEnvAttr routine enables the SQL/CLI application to set the value of a specific attribute. Attribute values may be set by the SQL/CLI 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 SQL/CLI 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 SQL/CLI application to determine the current value of a specific connection attribute. For connection attributes that may be set by the user, the SetConnectAttr routine enables the SQL/CLI application to set the value of a specific connection 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 connection attribute its name, code value, data type, possible values and whether the connection attribute may be set using SetConnectAttr.

The POPULATE IPD attribute determines whether the SQL/CLI implementation will populate the implementation parameter descriptor with an item descriptor area for each <dynamic parameter specification> when an

SQL-statement is prepared or executed immediately. The POPULATE IPD attribute is automatically set each time an SQL-connection is established for the allocated SQL-connection.

The SAVEPOINT NAME connection attribute specifies the savepoint to be referenced in an invocation of the EndTran routine that uses the SAVEPOINT NAME ROLLBACK or SAVEPOINT NAME RELEASE CompletionType, respectively. The SAVEPOINT NAME attribute is set to a zero-length string 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 SQL/CLI application to determine the current value of a specific statement attribute. For statement attributes that may be set by the user, the SetStmtAttr routine enables the SQL/CLI application to set the value of a specific statement 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 statement attribute its name, code value, data type, possible values, and whether the statement attribute may be set by using SetStmtAttr.

The APD HANDLE statement attribute is the value of the handle of the current application parameter descriptor for the allocated SQL-statement. The statement 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 statement attribute is the value of the handle of the current application row descriptor for the allocated SQL-statement. The statement 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 statement attribute is the value of the handle of the implementation parameter descriptor associated with the allocated SQL-statement. The statement 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 statement attribute is the value of the handle of the implementation row descriptor associated with the allocated SQL-statement. The statement 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 statement attribute determines the *scrollability* implicitly declared when Execute or ExecDirect are invoked. The statement attribute is set to NONSCROLLABLE when the SQL-statement is allocated. The CURSOR SENSITIVITY statement attribute determines the *sensitivity* to changes of the cursor implicitly declared when Execute or ExecDirect are invoked. The statement attribute is set to ASENSITIVE when the SQL-statement is allocated.

The CURSOR HOLDABLE statement attribute determines the *holdability* of the cursor implicitly declared when Execute or ExecDirect are invoked. The statement 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 an SQL-statement is initially allocated. It is reset to 1 (one) whenever Fetch or FetchScroll are successfully executed.

The NEST DESCRIPTOR statement attribute determines whether nested descriptor items are permitted in a CLI descriptor. Nested descriptor items are used to describe ROW, ARRAY, and MULTiset data types. The statement attribute is set to FALSE when the SQL-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, result columns of <dynamic select statement>s and <dynamic single row select statement>s, or <target specification>s for the result columns.

Each descriptor area comprises *header fields* and zero or more *item descriptor areas*. The header fields are specified in Table 5, “Fields in SQL/CLI row and parameter descriptor areas”. The header fields include a COUNT field that indicates the number of item descriptor areas and an ALLOC_TYPE field that indicates whether the CLI descriptor area was allocated by the user or automatically allocated by the SQL/CLI 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 following “Status record fields” in Table 5, “Fields in SQL/CLI row and parameter 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. IPDs and IRDs are collectively known as *implementation descriptor areas*; APDs and ARDs are collectively known as *application descriptor areas*.

When an SQL-statement is allocated, a CLI descriptor area of each type is automatically allocated by the SQL/CLI implementation. The ALLOC_TYPE fields for these CLI descriptor areas are set to indicate AUTOMATIC. A CLI descriptor area allocated by the user has its ALLOC_TYPE field 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 SQL/CLI 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.

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 that 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

This Subclause modifies Subclause 4.39, “Client-server operation”, in ISO/IEC 9075-2.

Insert this 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.

This page intentionally left blank.

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
| 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, M, 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) There shall be no <separator> between the <CLI name prefix> and the <CLI generic name>.
- 4) 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

```

- 5) *HL* shall support either the invocation of *CF* or the invocation of *CP*. It is implementation-defined which is supported.
- 6) 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.
- 7) 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.

- 8) 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 2, “Supported calling conventions of SQL/CLI routines by language”.

Table 2 — 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
M (ISO/IEC 11756)	Optional	Required
Pascal (ISO/IEC 7185 and ISO/IEC 10206)	Optional	Required
PL/I (ISO 6160)	Optional	Required

- 9) 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>.
- 10) 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.
- 11) 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:
- Any <CLI by-value prefix> remains unchanged.
 - Any <CLI by-reference prefix> is replaced by SQR.
 - The <CLI generic name> is replaced by an abbreviated version according to Table 3, “Abbreviated SQL/CLI generic names”.

Table 3 — Abbreviated SQL/CLI generic names

Generic Name	Abbreviation
AllocConnect	AC

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

Generic Name	Abbreviation
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
GetTypeInfo	GTI
MoreResults	MR
NextResult	NR
NumResultCols	NRC
ParamData	PRD
Prepare	PR
PrimaryKeys	PK

Generic Name	Abbreviation
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

12) Let CR be a <CLI routine> and let RN be its <CLI routine name>. Let RNU be the value of $UPPER(RN)$.

Case:

- If HL supports case sensitive routine names, then the name used for the invocation of CR shall be RN .
- If HL does not support <simple Latin lower case letter>s, then the name used for the invocation of CR shall be RNU .
- If HL does not support case sensitive routine names, then the name used for the invocation of CR shall be RN or RNU .

13) Let *operative data type correspondence table* be the data type correspondence table for HL as specified in [Subclause 5.15, “SQL/CLI data type correspondences”](#). Refer to the two columns of the operative data type correspondence table as the “SQL data type column” and the “host data type column”.

14) 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.

- If TS is “None”, then let $TS = TI$.
- 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”](#).

Conformance Rules

- 1) Without Feature C001, “CLI routine invocation in Ada”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in Ada.
- 2) Without Feature C002, “CLI routine invocation in C”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in C.
- 3) Without Feature C003, “CLI routine invocation in COBOL”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in COBOL.
- 4) Without Feature C004, “CLI routine invocation in Fortran”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in Fortran.
- 5) Without Feature C005, “CLI routine invocation in MUMPS”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in M.
- 6) Without Feature C006, “CLI routine invocation in Pascal”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in Pascal.

- 7) Without Feature C007, “CLI routine invocation in PL/I”, a conforming SQL/CLI application shall not contain an invocation of a <CLI routine> written in PL/I.

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 *RNM* be the <CLI routine name> of the <CLI routine> invoked by the host program and let *RN* be the SQL/CLI routine identified by *RNM*. 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 is not a value of the data type of the parameter, or if the value of any output argument resulting from the execution of the <CLI routine> is not a value supported by the SQL/CLI application 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 SQL/CLI application:
 - 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 SQL-connection, then the General Rules of Subclause 5.3, “Implicit set connection”, are applied to *EC* as the dormant SQL-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 SQL-connection, then the General Rules of Subclause 5.3, “Implicit set connection”, are applied to *EC* as the dormant SQL-connection.
 - 6) *RN* is invoked.
- iii) Otherwise, *RN* is invoked.
- 4) Case:
 - a) If *RN* is a CLI function, then:
 - i) The values of all output arguments are established.
 - ii) Let *RC* be the return value.
 - b) If *RN* 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 nor 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.

5.3 Implicit set connection

Function

Specify the rules for an implicit SET CONNECTION statement.

General Rules

- 1) Let *DC* be a dormant SQL-connection specified in an application of this Subclause.
- 2) If an SQL-transaction is active for the current SQL-connection and the SQL-implementation does not support transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported — multiple server transactions*.
- 3) If *DC* cannot be selected, then an exception condition is raised: *connection exception — connection failure*.
- 4) The current SQL-connection *CC* and current SQL-session become a dormant SQL-connection and a dormant SQL-session, respectively. The SQL-session context for *CC* is preserved and is not affected in any way by operations performed over the selected SQL-connection.

NOTE 4 — The SQL-session context is defined in Subclause 4.37, “SQL-sessions”, in ISO/IEC 9075-2.

- 5) *DC* becomes the current SQL-connection and the SQL-session associated with *DC* becomes the *current SQL-session*. The SQL-session context is restored to the same state as at the time *DC* became dormant.

NOTE 5 — The SQL-session context information is defined in Subclause 4.37, “SQL-sessions”, in ISO/IEC 9075-2.

- 6) The SQL-server for the subsequent execution of SQL-statements via CLI routine invocations is set to that of the current SQL-connection.

5.4 Implicit cursor

Function

Specify the rules for an implicit DECLARE CURSOR and OPEN statement.

General Rules

- 1) Let *SS* and *AS* be a *SELECT SOURCE* and *ALLOCATED STATEMENT* specified in an application of this Subclause.
- 2) If there is no cursor associated with *AS*, then a cursor is associated with *AS* and the cursor name associated with *AS* becomes the name of the cursor.
- 3) The General Rules of Subclause 5.6, “Implicit EXECUTE USING and OPEN USING clauses”, are applied to 'OPEN', *SS*, and *AS* as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.
- 4) If the value of the CURSOR SCROLLABLE attribute of *AS* is SCROLLABLE, then let *CT* be 'SCROLL'; otherwise, let *CT* be an empty string.
- 5) Case:
 - a) If the value of the CURSOR SENSITIVITY attribute of *AS* is INSENSITIVE, then let *CS* be 'INSENSITIVE'.
 - b) If the value of the CURSOR SENSITIVITY attribute of *AS* is SENSITIVE, then let *CS* be 'SENSITIVE'.
 - c) Otherwise, let *CS* be 'ASENSITIVE'.
- 6) If the value of the CURSOR HOLDABLE attribute of *AS* is HOLDABLE, then let *CH* be 'WITH HOLD'; otherwise, let *CH* be an empty string.
- 7) Let *CN* be the name of the cursor associated with *AS* and let *CR* be the following <declare cursor>:


```
DECLARE CN CS CT CURSOR CH FOR SS
```
- 8) Cursor *CN* is opened in the following steps:
 - a) A copy of *SS* is effectively created in which:
 - i) Each <dynamic parameter specification> is replaced by the value of the corresponding dynamic parameter.
 - ii) Each <value specification> generally contained in *SS* that is CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, CURRENT_PATH, CURRENT_DEFAULT_TRANSFORM_GROUP, or CURRENT_TRANSFORM_GROUP_FOR_TYPE <path-resolved user-defined type name> is replaced by the value resulting from evaluation of CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, 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.

- iii) Each <datetime value function> generally contained in *SS* is replaced by the value resulting from evaluation of that <datetime value function>, with all such evaluations effectively done at the same instant in time.
- b) Let *T* be the table specified by the copy of *SS*.
- c) A table descriptor for *T* is effectively created.
- d) The General Rules of Subclause 14.1, “<declare cursor>”, in ISO/IEC 9075-2 are applied to *CR*.
- e) Case:
 - i) If *CR* specifies *INSENSITIVE*, then a copy of *T* is effectively created and cursor *CN* is placed in the open state and its position is before the first row of the copy of *T*.
 - ii) Otherwise, cursor *CN* is placed in the open state and its position is before the first row of *T*.
- 9) If *CR* specifies *INSENSITIVE*, and the SQL-implementation is unable to guarantee that significant changes will be invisible through *CR* during the SQL-transaction in which *CR* is opened and every subsequent SQL-transaction during which it may be held open, then an exception condition is raised: *cursor sensitivity exception — request rejected*.
- 10) If *CR* specifies *SENSITIVE*, and the SQL-implementation is unable to guarantee that significant changes will be visible through *CR* during the SQL-transaction in which *CR* is opened, then an exception condition is raised: *cursor sensitivity exception — request rejected*.

NOTE 6 — The visibility of significant changes through a sensitive holdable cursor during a subsequent SQL-transaction is implementation-defined.
- 11) Whether an implementation is able to disallow significant changes that would not be visible through a currently open cursor is implementation-defined.

5.5 Implicit DESCRIBE USING clause

Function

Specify the rules for an implicit DESCRIBE USING clause.

General Rules

- 1) Let *S* and *AS* be a *SOURCE* and an *ALLOCATED STATEMENT* specified in the rules of this Subclause.
- 2) Let *IRD* and *IPD* be the implementation row descriptor and implementation parameter descriptor, respectively, associated with *AS*.
- 3) Let *HL* be the standard programming language of the invoking host program.
- 4) The value of *DYNAMIC_FUNCTION* and *DYNAMIC_FUNCTION_CODE* in *IRD* and *IPD* are respectively a character string representation of the prepared statement and a numeric code that identifies the type of the prepared statement.
- 5) A representation of the column descriptors of the <select list> columns for the prepared statement is stored in *IRD* as follows:
 - a) Case:
 - i) If there is a select source associated with *AS*, then:
 - 1) Let *TBL* be the table defined by *S* and let *D* be the degree of *TBL*.
Case:
 - A) If the value of the statement attribute *NEST_DESCRIPTOR* is *True*, then let *NS_i*, 1 (one) $\leq i \leq D$, be the number of subordinate descriptors of the descriptor for the *i*-th column of *T*.
 - B) Otherwise, let *NS_i*, 1 (one) $\leq i \leq D$, be 0 (zero).
 - 2) *TOP_LEVEL_COUNT* is set to *D*. If *D* is 0 (zero), then let *TD* be 0 (zero); otherwise, let *TD* be $D + \sum_{i=1}^D (NS_i)$. *COUNT* is set to *TD*.
 - 3) Let *SL* be the collection of <select list> columns of *TBL*.
 - 4) Case:
 - A) If some subset of *SL* is the primary key of *TBL*, then *KEY_TYPE* is set to 1 (one).
 - B) If some subset of *SL* is the preferred key of *TBL*, then *KEY_TYPE* is set to 2.
 - C) Otherwise, *KEY_TYPE* is set to 0 (zero).
 - ii) Otherwise:
 - 1) Let *D* be 0 (zero). Let *TD* be 0 (zero).

- 2) KEY_TYPE is set to 0 (zero).
- b) If *TD* is zero, then no item descriptor areas are set. Otherwise, the first *TD* item descriptor areas are set so that the *i*-th item descriptor area contains the descriptor of the *j*-th column of *TBL* such that:
- i) The descriptor for the first such column is assigned to the first descriptor area.
 - ii) The descriptor for the *j*+1-th column is assigned to the *i*+*NS_j*+1-th item descriptor area.
 - iii) If the value of the statement attribute NEST DESCRIPTOR is *True*, then the implicitly ordered subordinate descriptors for the *j*-th column are assigned to contiguous item descriptor areas starting at the *i*+1-th item descriptor area.
- c) The descriptor of a column consists of values for LEVEL, TYPE, NULLABLE, NAME, UNNAMED, KEY_MEMBER, and other fields depending on the value of TYPE as described below. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values. The DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields are not relevant in this case.
- i) If the item descriptor area is set to a descriptor that is immediately subordinate to another whose LEVEL value is some value *k*, then LEVEL is set to *k*+1; otherwise, LEVEL is set to 0 (zero).
 - ii) TYPE is set to a code as shown in Table 6, “Codes used for implementation data types in SQL/CLI”, indicating the data type of the column or subordinate descriptor.
 - iii) Case:
 - 1) If the value of LEVEL is 0 (zero), then:
 - A) If the resulting column is possibly nullable, then NULLABLE is set to 1 (one); otherwise NULLABLE is set to 0 (zero).
 - B) If the column name is implementation-dependent, then NAME is set to the implementation-dependent name of the column and UNNAMED is set to 1 (one); otherwise, NAME is set to the <derived column> name for the column and UNNAMED is set to 0 (zero).
 - C) Case:
 - I) If a <select list> column *C* is a member of a primary or preferred key of *TBL*, then KEY_MEMBER is set to 1 (one).
 - II) Otherwise, KEY_MEMBER is set to 0 (zero).
 - 2) Otherwise:
 - A) NULLABLE is set to 1 (one).
 - B) Case:
 - I) If the item descriptor area describes a field of a row type, then
 - Case:
 - 1) If the name of the field is implementation-dependent, then NAME is set to the implementation-dependent name of the field and UNNAMED is set to 1 (one).

- 2) Otherwise, NAME is set to the name of the field and UNNAMED is set to 0 (zero).
- II) Otherwise, UNNAMED is set to 1 (one) and NAME is set to an implementation-dependent value.
- C) KEY_MEMBER is set to 0 (zero).
- iv) Case:
 - 1) If TYPE indicates a <character string type>, then LENGTH is set to the length or maximum length in characters of the character string. OCTET_LENGTH is set to the maximum possible length in octets of the character string. If *HL* is C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string. CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set. COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string's collation.
 - 2) If TYPE indicates a <binary large object string type>, then LENGTH and OCTET_LENGTH are both set to the maximum length in octets of the binary large object 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”, to indicate the specific datetime data type, and PRECISION is set to the <time precision> or <timestamp precision> as 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”, 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 for the user-defined type.