

Information technology — Database languages — SQL

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

ICS 35.060

National foreword

This British Standard reproduces verbatim ISO/IEC 9075-3 : 1995 and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee IST/21, Open Systems Interconnection, Data Management and Open Distributed Processing, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled 'International Standards Correspondence Index', or using the 'Find' facility of the BSI Standards Electronic Catalogue.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, the ISO/IEC title page, pages ii to viii, pages 1 to 225 and a back cover.

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Board and comes into effect on 15 March 1997

© BSI 1997

ISBN 0 580 26588 9

Amendments issued since publication

Amd. No.	Date	Text affected

INTERNATIONAL STANDARD

**ISO/IEC
9075-3**

First edition
1995-12-15

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 9075-3:1995(E)



Documents

Document Sharing Hub

Contents	Page
1 Scope	1
2 Normative references	3
3 Definitions, notations, and conventions	5
3.1 Definitions	5
3.2 Notations	5
3.3 Conventions	5
3.3.1 Specification of routine definitions	5
3.3.2 Subclause naming	6
4 Concepts	7
4.1 Introduction	7
4.2 Return codes	9
4.3 Diagnostics areas	10
4.4 Miscellaneous characteristics	12
4.4.1 Handles	12
4.4.2 Null terminated strings	12
4.4.3 Null pointers	12
4.4.4 Environment attributes	13
4.4.5 Connection attributes	13
4.4.6 Statement attributes	13
4.4.7 CLI descriptor areas	14
5 Call-Level Interface specifications	17
5.1 <CLI routine>	17
5.2 <CLI routine> invocation	23
5.3 SQL/CLI common elements	26
5.3.1 Implicit set connection	26
5.3.2 Implicit cursor	26
5.3.3 Implicit using clause	27
5.3.4 Character string retrieval	34
5.3.5 Deferred parameter check	35
5.3.6 Client-server operation	35
5.3.7 CLI-specific status codes	35
5.3.8 Description of CLI item descriptor areas	37
5.3.9 Other tables associated with CLI	42
5.4 Data type correspondences	58

Descriptors: data processing, text processing, factual data bases, artificial languages, database languages, SQL, interfaces, computer interfaces.

ii Call-Level Interface (SQL/CLI)

6	SQL/CLI routines	65
6.1	AllocConnect	65
6.2	AllocEnv	66
6.3	AllocHandle	67
6.4	AllocStmt	70
6.5	BindCol	71
6.6	BindParam	73
6.7	Cancel	77
6.8	CloseCursor	79
6.9	ColAttribute	80
6.10	Connect	82
6.11	CopyDesc	86
6.12	DataSources	87
6.13	DescribeCol	89
6.14	Disconnect	91
6.15	EndTran	93
6.16	Error	96
6.17	ExecDirect	98
6.18	Execute	101
6.19	Fetch	103
6.20	FetchScroll	105
6.21	FreeConnect	108
6.22	FreeEnv	109
6.23	FreeHandle	110
6.24	FreeStmt	113
6.25	GetConnectAttr	114
6.26	GetCursorName	115
6.27	GetData	116
6.28	GetDescField	121
6.29	GetDescRec	123
6.30	GetDiagField	125
6.31	GetDiagRec	131
6.32	GetEnvAttr	133
6.33	GetFunctions	134
6.34	GetInfo	135
6.35	GetStmtAttr	143
6.36	GetTypeInfo	145
6.37	NumResultCols	149
6.38	ParamData	150
6.39	Prepare	155
6.40	PutData	157
6.41	RowCount	159
6.42	SetConnectAttr	160
6.43	SetCursorName	161
6.44	SetDescField	163

6.45	SetDescRec	166
6.46	SetEnvAttr	168
6.47	SetStmtAttr	169
7	Conformance	173
7.1	Introduction	173
7.2	Claims of conformance	173
7.3	Extensions and options	174
Annex A	Typical header files	175
A.1	C Header File SQLCLI.H	175
A.2	COBOL Library Item SQLCLI	185
Annex B	Sample C programs	193
B.1	Create table, insert, select	193
B.2	Interactive Query	196
B.3	Providing long dynamic arguments at Execute() time	199
Annex C	Implementation-defined elements	203
Annex D	Implementation-dependent elements	211
Index		215

TABLES

Table		Page
1	Fields in CLI diagnostics areas	11
2	Supported calling conventions of CLI routines by language	19
3	Abbreviated CLI generic names	20
4	SQLSTATE class and subclass values for CLI-specific conditions	35
5	Fields in CLI item descriptor areas	40
6	Codes used for implementation data types in CLI	41
7	Codes used for application data types in CLI	41
8	Codes associated with datetime data types in SQL/CLI	42
9	Codes associated with <interval qualifier> in SQL/CLI	42
10	SQL-statement integer codes for use in a diagnostics area	42
11	SQL-statement character codes for use in a diagnostics area	44
12	Codes used for diagnostic fields	45
13	Codes used for handle types	46
14	Codes used for transaction termination	46
15	Codes used for environment attributes	46
16	Codes used for connection attributes	46
17	Codes used for statement attributes	46
18	Codes used for FreeStmt options	46
19	Data types of attributes	47
20	Codes used for descriptor fields	48
21	Codes used for fetch orientation	49
22	Miscellaneous codes used in CLI	49
23	Codes used for GetData data types	49
24	Codes used to identify SQL/CLI routines	49
25	Codes and data types for implementation information	51
26	Values for ALTER TABLE with GetInfo	52
27	Values for CURSOR COMMIT BEHAVIOR with GetInfo	52
28	Values for FETCH DIRECTION with GetInfo	53
29	Values for GETDATA EXTENSIONS with GetInfo	53
30	Values for IDENTIFIER CASE with GetInfo	53
31	Values for OUTER JOIN CAPABILITIES with GetInfo	53
32	Values for SCROLL CONCURRENCY with GetInfo	54
33	Values for TRANSACTION CAPABLE with GetInfo	54
34	Values for TRANSACTION ISOLATION OPTION with GetInfo	54
35	Values for NULL COLLATION with GetInfo	54
36	Codes used for concise data types	55
37	Codes used with concise datetime data types in SQL/CLI	56
38	Codes used with concise interval data types in SQL/CLI	56

39 Concise codes used with datetime data types in SQL/CLI 57

40 Concise codes used with interval data types in SQL/CLI 57

41 Data type correspondences for Ada 58

42 Data type correspondences for C 59

43 Data type correspondences for COBOL 60

44 Data type correspondences for Fortran 61

45 Data type correspondences for MUMPS 62

46 Data type correspondences for Pascal 63

47 Data type correspondences for PL/I 64

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 21, *Open systems interconnection, data management and open distributed processing*.

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

- *Part 3: Call-Level Interface (SQL/CLI)*
- *Part 4: Persistent Stored Modules (SQL/PSM)*

Parts 1 and 2 are currently published as ISO/IEC 9075:1992.

Annexes A to D 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, “Conformance”, defines the criteria for conformance to this part of ISO/IEC 9075.
- 8) Annex A, “Typical header files”, is an informative Annex. It provides examples of typical header files for application programs using the SQL Call-Level Interface.
- 9) Annex B, “Sample C programs”, is an informative Annex. It provides a sample of using the SQL Call-Level Interface from the C programming language.
- 10) Annex C, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this part of the standard states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementor shall provide in each case.
- 11) Annex D, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this part of the standard states that the syntax or meaning or effect on the database is partly or wholly implementation-dependent.

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 7, “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 Call-Level Interface (SQL/CLI)

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this part of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard 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 7185:1990, *Information technology — Programming languages — Pascal*.

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

NOTE — ISO 8652:1987 has been superseded by a new edition (ISO/IEC 8652:1995). However, when this part of ISO/IEC 9075 was under development, the previous edition was valid and this part of ISO/IEC 9075 is therefore based on that edition, which is listed below.

ISO 8652:1987, *Programming languages — Ada*.

ISO/IEC 9075:1992, *Information technology — Database languages — SQL*.

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*.

4 Call-Level Interface (SQL/CLI)

3 Definitions, notations, and conventions

3.1 Definitions

For the purposes of this part of ISO/IEC 9075, the definitions given in ISO/IEC 9075:1992 and the following definitions apply.

- a) **handle**: An opaque data value 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.

3.2 Notations

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 3.2, "Notation", of ISO/IEC 9075:1992.

3.3 Conventions

The conventions used in this part of ISO/IEC 9075 are identical to those described in Subclause 3.3, "Conventions", of ISO/IEC 9075:1992.

The contents of this part of ISO/IEC 9075 depend wholly on ISO/IEC 9075:1992. For example, the Syntax found in the Format portions of this part of ISO/IEC 9075 often uses symbols that are defined in ISO/IEC 9075:1992.

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 numerical 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 Conventions

3.3.2 Subclause naming

Clauses and Subclauses in this part of ISO/IEC 9075 that have names identical to Clauses or Subclauses in ISO/IEC 9075:1992 supplement the Clause or Subclause, respectively, in ISO/IEC 9075:1992, typically by replacing Format items or Rules or by providing new Format items or Rules.

Clauses and Subclauses in this part of ISO/IEC 9075 that have names that are not identical to Clauses or Subclauses in ISO/IEC 9075:1992 provide language specification particular to this part of ISO/IEC 9075.

6 Call-Level Interface (SQL/CLI)

4 Concepts

4.1 Introduction

The Call-Level Interface (SQL/CLI) is an alternative binding style for executing SQL statements comprising 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, and
- Obtain information about the implementation.

The AllocHandle routine allocates the resources to manage an SQL-environment, an SQL-connection, a CLI descriptor area, or SQL-statement processing. 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

4.1 Introduction

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 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 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 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. The application sets values in the CLI descriptor area for the description of dynamic parameter values and, regardless of whether automatic population is supported, in the CLI descriptor area for the description of dynamic parameters either explicitly using the routines SetDescField and SetDescRec or implicitly using the routine BindParam. The value of a dynamic 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 parameter values. The PutData routine is used to provide the deferred values. The PutData routine also enables the values of character string parameters to be provided in pieces.

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.

Values for unbound columns can be individually retrieved by using the GetData routine. The GetData routine also enables the values of character string columns to be retrieved 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.

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.

8 Call-Level Interface (SQL/CLI)

4.1 Introduction

An SQL-transaction is terminated by using the EndTran routine.

NOTE 1 – Neither a <commit statement> nor a <rollback 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 GetFunctions, GetInfo, 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 zero indicates Success . The CLI routine executed successfully.
- A value of 1 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*.

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

In addition to providing the return code, for all CLI routines other than 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

4.3 Diagnostics areas

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.

If multiple status records are generated, 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 1, "Fields in CLI diagnostics areas".

10 Call-Level Interface (SQL/CLI)

4.3 Diagnostics areas

Table 1—Fields in 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
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>)
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
SCHEMA_NAME	CHARACTER VARYING (<i>L</i>)
SERVER_NAME	CHARACTER VARYING (<i>L</i>)
SQLSTATE	CHARACTER (5)
SUBCLASS_ORIGIN	CHARACTER VARYING (<i>L1</i>)
TABLE_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.	

All diagnostics area fields specified in ISO/IEC 9075:1992 that are not included in this table are not applicable to SQL/CLI.

4.4 Miscellaneous characteristics

4.4 Miscellaneous characteristics

4.4.1 Handles

The AllocHandle routine returns an identifier, known as a *handle*, that uniquely identifies the allocated resource. Although the CLI parameter 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.

NOTE 3 – 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 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 application wishes to prohibit the return of this information.
- In lieu of other output arguments where specifically allowed by 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 ISO/IEC 9075. The semantics of such a specification depend on the context.

12 Call-Level Interface (SQL/CLI)

4.4 Miscellaneous characteristics

If the 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 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 5.3.9, “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 5.3.9, “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.

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 5.3.9, “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.

4.4 Miscellaneous characteristics

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 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 when the SQL-statement is allocated.

The CURSOR SCROLLABLE attribute determines the *scrollability* of the cursor 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 UNSPECIFIED 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. It consists of zero or more item descriptor areas, together with a COUNT field of data type SMALLINT that indicates the number of item descriptor areas and an ALLOC_TYPE field of data type SMALLINT that indicates whether the CLI descriptor area was allocated by the user or automatically allocated by the implementation. The COUNT field is set to 0 when the CLI descriptor area is allocated. Each CLI item descriptor area consists of the fields specified in Table 5, "Fields in CLI item descriptor areas", in Subclause 5.3.8, "Description of CLI item descriptor areas". The DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields are set to 0 when the CLI descriptor area is allocated and all the other fields in the CLI item descriptor areas are initially undefined.

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 BindParam 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.

14 Call-Level Interface (SQL/CLI)

4.4 Miscellaneous characteristics

NOTE 4 – 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.

16 Call-Level Interface (SQL/CLI)

5 Call-Level Interface specifications

5.1 <CLI routine>

Function

Describe a generic SQL/CLI routine.

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
    | BindParam
    | Cancel
    | CloseCursor
    | ColAttribute
    | Connect
    | CopyDesc
    | DataSources
    | DescribeCol
    | Disconnect
    | EndTran
    | Error
    | ExecDirect
    | Execute
    | Fetch
    | FetchScroll
    | FreeConnect
    | FreeEnv
    | FreeHandle
    | FreeStmt
    | GetConnectAttr
    | GetCursorName
    | GetData
    | GetDescField
    | GetDescRec

```

5.1 <CLI routine>

```

| GetDiagField
| GetDiagRec
| GetEnvAttr
| GetFunctions
| GetInfo
| GetStmtAttr
| GetTypeInfo
| NumResultCols
| ParamData
| Prepare
| PutData
| RowCount
| SetConnectAttr
| SetCursorName
| SetDescField
| SetDescRec
| SetEnvAttr
| SetStmtAttr
| <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 and 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*.

18 Call-Level Interface (SQL/CLI)

5.1 <CLI routine>

- 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 2, "Supported calling conventions of CLI routines by language".

Table 2—Supported calling conventions of 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 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>.

5.1 <CLI routine>

- 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 3, "Abbreviated CLI generic names".

Table 3—Abbreviated CLI generic names

Generic Name	Abbreviation
AllocConnect	AC
AllocEnv	AE
AllocHandle	AH
AllocStmt	AS
BindCol	BC
BindParam	BP
Cancel	CAN
CloseCursor	CC
ColAttribute	CO
Connect	CON
CopyDesc	CD
DataSources	DS
DescribeCol	DC
Disconnect	DIS
EndTran	ET
Error	ER
ExecDirect	ED
Execute	EX
Fetch	FT
FetchScroll	FTS
FreeConnect	FC
FreeEnv	FE
FreeHandle	FH

5.1 <CLI routine>

Table 3—Abbreviated CLI generic names (Cont.)

Generic Name	Abbreviation
FreeStmt	FS
GetCursorName	GCN
GetConnectAttr	GCA
GetData	GDA
GetDescField	GDF
GetDescRec	GDR
GetDiagField	GXF
GetDiagRec	GXR
GetEnvAttr	GEA
GetFunctions	GFU
GetInfo	GI
GetStmtAttr	GSA
GetTypeInfo	GTI
NumResultCols	NRC
ParamData	PRD
Prepare	PR
PutData	PTD
RowCount	RC
SetConnectAttr	SCA
SetCursorName	SCN
SetDescField	SDF
SetDescRec	SDR
SetEnvAttr	SEA
SetStmtAttr	SSA

- 11) 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*.

5.1 <CLI routine>

- 12) Let “operative data type correspondence table” be the data type correspondence table for *HL* as specified in Subclause 5.4, “Data type correspondences”, Refer to the two columns of the operative data type correspondence table as the “SQL data type column” and the “host data type column”.
- 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 the <CLI routine> are specified in Subclause 5.2, “<CLI routine> invocation”.

22 Call-Level Interface (SQL/CLI)

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 5.3.1, "Implicit set connection", are applied to *EC* as the dormant connection.

5.2 <CLI routine> invocation

- 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 5.3.1, "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*.

24 Call-Level Interface (SQL/CLI)

5.2 <CLI routine> invocation

- 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 International Standard 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 '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".
 - b) Otherwise, no diagnostics area is updated.

5.3 SQL/CLI common elements

5.3 SQL/CLI common elements

5.3.1 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 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 and current SQL-session become a dormant SQL-connection and a dormant SQL-session, respectively. The SQL-session context information is preserved and is not affected in any way by operations performed over the selected SQL-connection.
NOTE 5 – The SQL-session context information is defined in Subclause 4.30, "SQL-sessions", in ISO/IEC 9075:1992.
- 5) *DC* becomes the current SQL-connection and the SQL-session associated with *DC* becomes the current SQL-session. All SQL-session context information is restored to the same state as at the time *DC* became dormant.
NOTE 6 – The SQL-session context information is defined in Subclause 4.30, "SQL-sessions", in ISO/IEC 9075:1992.
- 6) The SQL-server for the subsequent execution of SQL-statements via CLI routine invocations is set to that of the current SQL-connection.

5.3.2 Implicit cursor

Function

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

General Rules

- 1) Let *SS* and *AS* be a *SELECT SOURCE* and *ALLOCATED STATEMENT* specified in an application of this Subclause.
- 2) If there is no cursor associated with *AS*, then a cursor is associated with *AS* and the cursor name associated with *AS* becomes the name of the cursor.
- 3) The General Rules of Subclause 5.3.3, "Implicit using clause", 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.

26 Call-Level Interface (SQL/CLI)

5.3 SQL/CLI common elements

5) Case:

- a) If the value of the `CURSOR SENSITIVITY` attribute of *AS* is `INSENSITIVE`, then let *CS* be `INSENSITIVE`.
- b) Otherwise, let *CS* be an empty string.

6) Let *CN* be the name of the cursor associated with *AS* and let *CR* be the following <declare cursor>:

```
DECLARE CN CS CT CURSOR FOR SS
```

7) 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 `USER`, `CURRENT_USER`, `SESSION_USER` or `SYSTEM_USER` is replaced by the value resulting from evaluation of `USER`, `CURRENT_USER`, `SESSION_USER`, or `SYSTEM_USER`, respectively, with all such evaluations effectively done at the same instant in time; and
 - 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 13.1, "<declare cursor>", in ISO/IEC 9075:1992, 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*.

5.3.3 Implicit using clause

Function

Define the input/output variables for SQL-statement processing.

General Rules

- 1) Let *T*, *S*, and *AS* be a *TYPE*, *SOURCE*, and *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*.

5.3 SQL/CLI common elements

- 3) Let *HL* be the standard programming language of the invoking host program.
- 4) If *T* is 'DESCRIBE', then a representation of the column descriptors of the <select list> columns for the prepared statement is stored in *IRD* as follows:
 - a) If there is a select source associated with *AS*, then let *TBL* be the table defined by *S* and let *D* be the degree of *TBL*. Otherwise, let *D* be 0.
 - b) COUNT is set to *D*.
 - c) If *D* is zero, then no item descriptor areas are set. Otherwise, the first *D* item descriptor areas are set so that the *i*-th item descriptor area contains the descriptor of the *i*-th column of *TBL*. The descriptor of a column consists of values for TYPE, NULLABLE, NAME, UNNAMED, 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) TYPE is set to a code as shown in Table 6, "Codes used for implementation data types in CLI", indicating the data type of the column.
 - ii) If the resulting column is possibly nullable, then NULLABLE is set to 1; otherwise NULLABLE is set to 0.
 - iii) 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; otherwise, NAME is set to the <derived column> name for the column and UNNAMED is set to 0.
 - 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 the maximum possible length in octets of the character string. If *HL* is C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string. CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set. COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string's collation.
 - 2) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.
 - 3) If TYPE indicates 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> if either is applicable.

5.3 SQL/CLI common elements

- 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.
- 5) Let *C* be the allocated SQL-connection with which *AS* is associated.
- 6) If *T* is ‘DESCRIBE’ and POPULATE IPD for *C* is *false*, then no further rules of this Subclause are applied.
- 7) If *T* is ‘DESCRIBE’ and POPULATE IPD for *C* is *true*, then a descriptor for the <dynamic parameter specification>s for the prepared statement is stored in *IPD* as follows:
 - a) Let *D* be the number of <dynamic parameter specification>s in *S*.
 - b) COUNT is set to *D*.
 - c) If *D* is zero, then no item descriptor areas are set. Otherwise, the first *D* item descriptor areas are set so that the *i*-th item descriptor area contains a descriptor of the *i*-th <dynamic parameter specification>. The descriptor of a <dynamic parameter specification> consists of values for TYPE, NULLABLE, NAME, UNNAMED, PARAMETER_MODE, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, PARAMETER_SPECIFIC_NAME, and other fields depending on the value of TYPE as described below. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values. The DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields are not relevant in this case.
 - i) TYPE is set to a code as shown in Table 6, “Codes used for implementation data types in CLI”, indicating the data type of the <dynamic parameter specification>.
 - ii) NULLABLE is set to 1.
NOTE 7 – This indicates that the <dynamic parameter specification> can have the null value.
 - iii) UNNAMED is set to 1 and NAME is set to an implementation-dependent value.
 - 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 the maximum possible length in octets of the character string. If *HL* is C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string. CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the <character set name> of the character string’s character set. COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are set to the <collation name> of the character string’s collation.
 - 2) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.
 - 3) If TYPE indicates an <exact numeric type>, then PRECISION and SCALE are set to the precision and scale of the exact numeric.

5.3 SQL/CLI common elements

- 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> if either is applicable.
 - 6) If TYPE indicates INTERVAL, then LENGTH is set to the length in positions of the interval type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 9, “Codes associated with <interval qualifier> in SQL/CLI”, to indicate the specific <interval qualifier>, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and PRECISION is set to the <interval fractional seconds precision>, if applicable.
- 8) Let *ARD* and *APD* be the current application row descriptor and current application parameter descriptor, respectively, for *AS*.
 - 9) If *T* is 'EXECUTE' or 'OPEN', then *IPD* and *APD* describe the <dynamic parameter specification>s and <dynamic parameter specification> values, respectively, for the statement being executed. Let *D* be the number of <dynamic parameter specification>s in *S*.
 - a) If the value of COUNT for *APD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.
 - b) If the value of COUNT for *IPD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.
 - c) If the value of COUNT for *APD* is less than the value of COUNT for *IPD*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.
 - d) If the value of COUNT for *IPD* is less than *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.
 - e) If the value of COUNT for *IPD* is greater than *D*, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.
 - f) If the first *D* item descriptor areas of *IPD* are not valid as specified in Subclause 5.3.8, “Description of CLI item descriptor areas”, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.
 - g) For each of the first *D* item descriptor areas of *APD*, if TYPE indicates DEFAULT, then:
 - i) Let *TP*, *P*, and *SC* be the values of TYPE, PRECISION, and SCALE, respectively, for the corresponding item descriptor area of *IPD*.
 - ii) The data type, precision, and scale of the described <dynamic parameter specification> value are set to *TP*, *P*, and *SC*, respectively, for the purposes of this invocation only.
 - h) If the first *D* item descriptor areas of *APD* are not valid as specified in Subclause 5.3.8, “Description of CLI item descriptor areas”, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

30 Call-Level Interface (SQL/CLI)

5.3 SQL/CLI common elements

- i) For each item descriptor area for which DEFERRED is false in the first *D* item descriptor areas of *APD*, refer to the corresponding <dynamic parameter specification> value as an immediate parameter value and refer to the corresponding <dynamic parameter specification> as an immediate parameter.
- j) For the *i*-th immediate parameter value as described by the corresponding item descriptor area *IDA* of *APD*:
 - i) If NULL is false for *IDA*, then:
 - 1) Let *V* be the value of the host variable addressed by DATA_POINTER.
 - 2) Case:
 - A) If TYPE indicates CHARACTER, then
 - Case:
 - I) If OCTET_LENGTH_POINTER is zero or if OCTET_LENGTH_POINTER is not zero and the value of the host variable addressed by OCTET_LENGTH_POINTER indicates NULL TERMINATED, then let *L* be the number of characters of *V* that precede the implementation-defined null character that terminates a C character string.
 - II) Otherwise, let *Q* be the value of the host variable addressed by OCTET_LENGTH_POINTER and let *L* be the number of characters wholly contained in the first *Q* octets of *V*.
 - B) Otherwise, let *L* be zero.
 - 3) Let *SV* be *V* with effective data type *SDT*, as represented by the length value *L* and by the values of TYPE, PRECISION, and SCALE.
 - ii) Otherwise, let *SV* be the null value.
 - k) Let *TDT* be the effective data type of the *i*-th immediate parameter as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME in the corresponding item descriptor area of *IPD*.
 - l) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the Syntax Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.
 - m) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the General Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992, then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992.

5.3 SQL/CLI common elements

- n) Let *TV* be the value obtained, with data type *TDT*, by effectively performing the <cast specification>

CAST (*SV* AS *TDT*)

- o) Let *ADT* be the effective data type of the actual *i*-th immediate parameter, defined to be the data type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would automatically be set in the corresponding item descriptor area of *IPD* if POPULATE IPD was true for *C*.

- p) If the <cast specification>

CAST (*TV* AS *ADT*)

violates the Syntax Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

- q) If the <cast specification>

CAST (*TV* AS *ADT*)

violates the General Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992, then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992.

- r) The <cast specification>

CAST (*TV* AS *ADT*)

is effectively performed and is the value of the *i*-th immediate parameter.

- s) If DEFERRED is true for at least one of the first *D* item descriptor areas of *APD*, then:

- i) Let *PN* be the parameter number associated with the first such item descriptor area.
- ii) *PN* becomes the deferred parameter number associated with *AS*.
- iii) If *T* is 'EXECUTE', then *S* becomes the statement source associated with *AS*.
- iv) An exception condition is raised: *CLI-specific condition — dynamic parameter value needed*.

- 10) If *T* is 'FETCH', then *IRD* and *ARD* describe the <select list> columns and <target specification>s, respectively, for the column values that are to be retrieved. Let *D* be the degree of the table defined by *S*.

- a) If the value of COUNT for *ARD* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.
- b) Refer to a <target specification> whose corresponding item descriptor area has a non-zero DATA_POINTER as a *bound target* and refer to the corresponding <select list> column as a *bound column*.
- c) If every item descriptor area corresponding to a bound target in the first *D* item descriptor areas of *ARD* is not valid as specified in Subclause 5.3.8, "Description of CLI item descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

32 Call-Level Interface (SQL/CLI)

5.3 SQL/CLI common elements

- d) Let *SDT* be the effective data type of the *i*-th bound column as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME in the corresponding item descriptor area of *IRD*. Let *SV* be the value of the <select list> column, with data type *SDT*.
- e) Let *T1*, *OL*, *DP*, *IP*, and *LP* be the values of TYPE, OCTET_LENGTH, DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER, respectively, in the item descriptor area corresponding to the *i*-th bound target.
- f) Case:
- i) If *T1* indicates CHARACTER, then:
 - 1) Let *UT* be the code value corresponding to CHARACTER VARYING as specified in Table 6, "Codes used for implementation data types in CLI".
 - 2) Let *LV* be the implementation-defined maximum length for a CHARACTER VARYING data type.
 - ii) Otherwise, let *UT* be *T1* and let *LV* be 0.
- g) Let *TDT* be the effective data type of the *i*-th bound target as represented by the type *UT*, the length value *LV*, and the values of PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME in the corresponding item descriptor area of *ARD*.
- h) If the <cast specification>
- CAST (*SV* AS *TDT*)
- violates the Syntax Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.
- i) If the <cast specification>
- CAST (*SV* AS *TDT*)
- violates the General Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992, then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>", of ISO/IEC 9075:1992.
- j) The <cast specification>
- CAST (*SV* AS *TDT*)
- is effectively performed and the result is the value *TV* of the *i*-th bound target.
- k) If *TV* is the null value, then
- Case:
- i) If *IP* is zero, then an exception condition is raised: *data exception — null value, no indicator parameter*.
 - ii) Otherwise, the value of the host variable addressed by *IP* is set to -1.

5.3 SQL/CLI common elements

- 1) If *TV* is not the null value, then:
 - i) If *IP* is not zero, then the value of the host variable addressed by *IP* is set to 0.
 - ii) Case:
 - 1) If *T1* does not indicate CHARACTER, then the value of the host variable addressed by *DP* is set to *TV*.
 - 2) Otherwise the General Rules of Subclause 5.3.4, “Character string retrieval”, are applied with *DP*, *TV*, *OL*, and *LP* as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

5.3.4 Character string retrieval

Function

Specify the rules for retrieving character string values.

General Rules

- 1) Let *T*, *V*, *TL*, and *RL* be a *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH* specified in an application of this Subclause.
- 2) If *TL* is not greater than zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.
- 3) Let *L* be the length in octets of *V*.
- 4) If *RL* is not a null pointer, then *RL* is set to *L*.
- 5) Case:
 - a) If null termination is *false* for the current SQL-environment, then:
 - i) If *L* is not greater than *TL*, then the first *L* octets of *T* are set to *V* and the values of the remaining octets of *T* are implementation-dependent.
 - ii) Otherwise, *T* is set to the first *L* octets of *V* and a completion condition is raised: *warning — string data, right truncation*.
 - b) Otherwise, let *NB* be the length in octets of a null terminator in the character set of the *i*-th bound target.

Case:

 - i) If *L* is not greater than $(TL - NB)$, then the first $(L + NB)$ octets of *T* are set to *V* concatenated with a single implementation-defined null character that terminates a C character string. The values of the remaining characters of *T* are implementation-dependent.
 - ii) Otherwise, *T* is set to the first $(TL - NB)$ octets of *V* concatenated with a single implementation-defined null character that terminates a C character string and a completion condition is raised: *warning — string data, right truncation*.

34 Call-Level Interface (SQL/CLI)

5.3 SQL/CLI common elements

5.3.5 Deferred parameter check

Function

Check for the existence of deferred dynamic parameters when accessing a CLI descriptor.

General Rules

- 1) Let *DA* be a DESCRIPTOR AREA specified in an application of this Subclause.
- 2) Let *C* be the allocated SQL-connection with which *DA* is associated.
- 3) Let *L1* be a list of allocated SQL-statements associated with *C*.
- 4) Let *L2* be a list of allocated SQL-statements in *L1* which have an associated deferred parameter number.
- 5) Let *L3* be a list of CLI descriptor areas that are either the current application parameter descriptor for, or the implementation parameter descriptor associated with, an allocated SQL-statement in *L2*.
- 6) If *DA* is contained in *L3*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

5.3.6 Client-server operation

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.

5.3.7 CLI-specific status codes

Some of the conditions that can occur during the execution of CLI routines are CLI-specific. The corresponding status codes are listed in Table 4, “SQLSTATE class and subclass values for CLI-specific conditions”.

Table 4—SQLSTATE class and subclass values for CLI-specific conditions

Condition	Class	Subcondition	Subclass
CLI-specific condition	HY	(no subclass)	000
		associated statement is not prepared	007
		attempt to concatenate a null value	020
		attribute cannot be set now	011
		cannot modify an implementation row descriptor	016

5.3 SQL/CLI common elements

Table 4—SQLSTATE class and subclass values for CLI-specific conditions (Cont.)

Condition	Class	Subcondition	Subclass
		descriptor invalid on indirect reference	023
		dynamic parameter value needed	(See the Note at the end of the table)
		function sequence error	010
		inconsistent descriptor information	021
		invalid attribute identifier	092
		invalid attribute value	024
		invalid data type	004
		invalid data type in application descriptor	003
		invalid descriptor field identifier	091
		invalid fetch orientation	106
		invalid handle	(See the Note at the end of the table)
		invalid information type	096
		invalid LengthPrecision value	104
		invalid retrieval code	103
		invalid string length or buffer length	090
		invalid transaction operation code	012
		invalid use of automatically-allocated descriptor handle	017
		invalid use of null pointer	009
		limit on number of handles exceeded	014
		memory allocation error	001
		memory management error	013
		non-string data cannot be sent in pieces	019
		operation canceled	008
		optional feature not implemented	C00
		server declined the cancellation request	018

NOTE 8 – No subclass value is defined for the subcondition *invalid handle* since no diagnostic information can be generated in this case or for the subcondition *dynamic parameter value needed*, since no diagnostic information is generated in this case.

36 Call-Level Interface (SQL/CLI)

5.3 SQL/CLI common elements

5.3.8 Description of CLI item descriptor areas

Function

Specify the identifiers, data types and codes for fields used in CLI item descriptor areas.

Syntax Rules

- 1) A CLI item descriptor area comprises the fields specified in Table 5, "Fields in CLI item descriptor areas".
- 2) A CLI item descriptor area in an implementation parameter descriptor is *valid* if and only if TYPE is one of the code values in Table 6, "Codes used for implementation data types in CLI", and one of the following is true:
Case:
 - a) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.
 - b) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.
 - c) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type.
 - d) TYPE indicates INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.
 - e) TYPE indicates CHARACTER or CHARACTER VARYING and LENGTH is a valid length value for a <character string type>.
 - f) TYPE indicates BIT or BIT VARYING and LENGTH is a valid length value for <bit string type>.
 - g) TYPE indicates a <datetime type>, DATETIME_INTERVAL_CODE is one of the code values in Table 8, "Codes associated with datetime data types in SQL/CLI", and PRECISION is a valid value for the <time precision> or <timestamp precision> of the indicated datetime data type.
 - h) TYPE indicates an <interval type>, DATETIME_INTERVAL_CODE is one of the code values in Table 9, "Codes associated with <interval qualifier> in SQL/CLI", to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is a valid <interval leading precision>, and PRECISION is a valid value for <interval fractional seconds precision>, if applicable.
- 3) Let *HL* be the standard programming language of the invoking host program. Let *operative data type correspondence table* be the data type correspondence table for *HL* as specified in Subclause 5.4, "Data type correspondences", Refer to the two columns of the operative data type correspondence table as the *SQL data type column* and the *host data type column*.
- 4) A CLI item descriptor area in a CLI descriptor area that is neither an implementation row descriptor nor an implementation parameter descriptor is *consistent* if and only if:
 - a) TYPE indicates DEFAULT or is one of the code values in Table 7, "Codes used for application data types in CLI"; and

5.3 SQL/CLI common elements

- b) If TYPE is one of the code values in Table 7, "Codes used for application data types in CLI", then the row that contains the SQL data type corresponding to TYPE in the SQL data type column of the operative data type correspondence table does not contain "None" in the host data type column; and
- c) One of the following is true:
 - i) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type; or
 - ii) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type; or
 - iii) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type; or
 - iv) TYPE indicates DEFAULT, CHARACTER, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.
- 5) Let IDA be a CLI item descriptor area in an application parameter descriptor.
- 6) If OCTET_LENGTH_POINTER for IDA has the same non-zero value as INDICATOR_POINTER for IDA, then SHARE is true for IDA; otherwise SHARE is false for IDA.
- 7) Case:
 - a) If SHARE is true and the value of the commonly addressed host variable is -1, then NULL is true for IDA.
 - b) If SHARE is false, INDICATOR_POINTER is not zero, and the value of the host variable addressed by INDICATOR_POINTER is negative, then NULL is true for IDA.
 - c) Otherwise, NULL is false for IDA.
- 8) If NULL is false, OCTET_LENGTH_POINTER is not zero, and the value of the host variable addressed by OCTET_LENGTH_POINTER indicates DATA AT EXEC, then DEFERRED is true for IDA; otherwise DEFERRED is false for IDA.
- 9) IDA is *valid* if and only if:
 - a) TYPE is one of the code values in Table 7, "Codes used for application data types in CLI".
 - b) The row that contains the SQL data type corresponding to TYPE in the SQL data type column of the operative data type correspondence table does not contain 'None' in the host data type column.
 - c) One of the following is true:
 - Case:
 - i) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.
 - ii) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.

38 Call-Level Interface (SQL/CLI)

5.3 SQL/CLI common elements

- iii) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type.
 - iv) TYPE indicates INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.
 - v) TYPE indicates CHARACTER, and one of the following is true:
 - 1) NULL is true.
 - 2) DEFERRED is true.
 - 3) OCTET_LENGTH_POINTER is not zero, the value *V* of the host variable *HV* addressed by OCTET_LENGTH_POINTER is greater than zero, and the number of characters wholly contained in the first *V* octets of *HV* is a valid length value for a CHARACTER data type.
 - 4) OCTET_LENGTH_POINTER is not zero, the value of the host variable addressed by OCTET_LENGTH_POINTER indicates NULL TERMINATED, and the number of characters of the value of the host variable addressed by DATA_POINTER that precede the implementation-defined null character that terminates a C character string is a valid length value for a CHARACTER data type.
 - 5) OCTET_LENGTH_POINTER is zero and the number of characters of the value of the host variable addressed by DATA_POINTER that precede the implementation-defined null character that terminates a C character string is a valid length value for a CHARACTER data type.
 - d) One of the following is true:
 - i) DATA_POINTER is zero and NULL is true.
 - ii) DATA_POINTER is zero and DEFERRED is true.
 - iii) DATA_POINTER is not zero and one of the following is true:
 - 1) NULL is true.
 - 2) DEFERRED is true.
 - 3) The value of the host variable addressed by DATA_POINTER is a valid value of the data type indicated by TYPE.
- 10) A CLI item descriptor area in an application row descriptor is *valid* if and only if:
- a) TYPE is one of the code values in Table 7, "Codes used for application data types in CLI".
 - b) The row that contains the SQL data type corresponding to TYPE in the SQL data type column of the operative data type correspondence table does not contain 'None' in the host data type column.
 - c) One of the following is true:
Case:
 - i) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.

5.3 SQL/CLI common elements

- ii) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.
- iii) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type.
- iv) TYPE indicates CHARACTER, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

Table 5—Fields in CLI item descriptor areas

Field	Data Type
CHARACTER_SET_CATALOG	CHARACTER VARYING(L)
CHARACTER_SET_NAME	CHARACTER VARYING(L)
CHARACTER_SET_SCHEMA	CHARACTER VARYING(L)
COLLATION_CATALOG	CHARACTER VARYING(L)
COLLATION_NAME	CHARACTER VARYING(L)
COLLATION_SCHEMA	CHARACTER VARYING(L)
DATA_POINTER	host variable address
DATETIME_INTERVAL_CODE	SMALLINT
DATETIME_INTERVAL_PRECISION	SMALLINT
INDICATOR_POINTER	host variable address
LENGTH	INTEGER
NAME	CHARACTER VARYING(L)
NULLABLE	SMALLINT
OCTET_LENGTH	INTEGER
OCTET_LENGTH_POINTER	host variable address
PARAMETER_MODE	SMALLINT ¹
PARAMETER_ORDINAL_POSITION	SMALLINT ¹
PARAMETER_SPECIFIC_CATALOG	CHARACTER VARYING(L) ¹
PARAMETER_SPECIFIC_NAME	CHARACTER VARYING(L) ¹
PARAMETER_SPECIFIC_SCHEMA	CHARACTER VARYING(L) ¹
PRECISION	SMALLINT
SCALE	SMALLINT
TYPE	SMALLINT
UNNAMED	SMALLINT

¹These fields are defined in part 4 of ISO/IEC 9075
Where L is an implementation-defined integer not less than 128.

General Rules

- Table 6, “Codes used for implementation data types in CLI”, specifies the codes associated with the SQL data types used in implementation descriptor areas.

40 Call-Level Interface (SQL/CLI)