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)

# Contents

Page

# TABLES

# 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 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.

### 3.3.2 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.

# 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.

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

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 '02*nnn*', 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".

**Table 1—Fields in CLI diagnostics areas**

| Field | Data type |
|---|---|
| **Header fields** | |
| DYNAMIC_FUNCTION | CHARACTER VARYING (*L1*) |
| DYNAMIC_FUNCTION_CODE | INTEGER . |
| MORE | INTEGER |
| NUMBER | INTEGER |
| RETURNCODE | SMALLINT |
| ROW_COUNT | INTEGER |
| **Fields in status records** | |
| CATALOG_NAME | CHARACTER VARYING (*L*) |
| CLASS_ORIGIN | CHARACTER VARYING (*L1*) |
| COLUMN_NAME | 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 |
| SCHEMA_NAME | CHARACTER VARYING (*L*) |
| SERVER_NAME | CHARACTER VARYING (*L*) |
| SQLSTATE | CHARACTER (5) |
| SUBCLASS_ORIGIN | CHARACTER VARYING (*L1*) |
| TABLE_NAME | CHARACTER VARYING (*L*) |

**Where** *L* is an implementation-defined integer not less than 128 and *L1* 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

### 4.4.1 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.

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.

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.

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.

# 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*.

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>.

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 |

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:

a) If *HL* supports case sensitive routine names, then the name used for the invocation of *CR* shall be *RN*.

b) If *HL* does not support <simple Latin lower case letter>s, then the name used for the invocation of *CR* shall be *RNU*.

c) If *HL* does not support case sensitive routine names, then the name used for the invocation of *CR* shall be *RN* or *RNU*.

12) Let "operative data type correspondence table" be the data type correspondence table for *HL* as specified in Subclause 5.4, "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".

# 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.

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*.

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

## 5.3.1 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.

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*.

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.

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.

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*.

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.

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*.

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.

l) 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*.

### 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-statment 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 |

**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.

### 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

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.

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.

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 |

[1]These fields are defined in part 4 of ISO/IEC 9075

**Where** *L* is an implementation-defined integer not less than 128.



## General Rules

1) Table 6, "Codes used for implementation data types in CLI", specifies the codes associated with the SQL data types used in implementation descriptor areas.

**Table 6—Codes used for implementation data types in CLI**

| Data Type | Code |
|---|---|
| Implementation-defined data type | <0 |
| BIT | 14 |
| BIT VARYING | 15 |
| CHARACTER | 1 |
| CHARACTER VARYING | 12 |
| DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, or TIMESTAMP WITH TIME ZONE | 9 |
| DECIMAL | 3 |
| DOUBLE PRECISION | 8 |
| FLOAT | 6 |
| INTEGER | 4 |
| INTERVAL | 10 |
| NUMERIC | 2 |
| REAL | 7 |
| SMALLINT | 5 |

2) Table 7, "Codes used for application data types in CLI", specifies the codes associated with the SQL data types used in application descriptor areas.

**Table 7—Codes used for application data types in CLI**

| Data Type | Code |
|---|---|
| Implementation-defined data type | <0 |
| CHARACTER | 1 |
| DECIMAL | 3 |
| DOUBLE PRECISION | 8 |
| FLOAT | 6 |
| INTEGER | 4 |
| NUMERIC | 2 |
| REAL | 7 |
| SMALLINT | 5 |

3) Table 8, "Codes associated with datetime data types in SQL/CLI", specifies the codes associated with the datetime data types allowed in SQL/CLI.

**Table 8—Codes associated with datetime data types in SQL/CLI**

| Datetime Data Type | Code |
|---|---|
| DATE | 1 |
| TIME | 2 |
| TIME WITH TIME ZONE | 4 |
| TIMESTAMP | 3 |
| TIMESTAMP WITH TIME ZONE | 5 |

4) Table 9, "Codes associated with <interval qualifier> in SQL/CLI", specifies the codes associated with <interval qualifier>s for interval data types in SQL/CLI.

**Table 9—Codes associated with <interval qualifier> in SQL/CLI**

| Interval qualifier | Code |
|---|---|
| DAY | 3 |
| DAY TO HOUR | 8 |
| DAY TO MINUTE | 9 |
| DAY TO SECOND | 10 |
| HOUR | 4 |
| HOUR TO MINUTE | 11 |
| HOUR TO SECOND | 12 |
| MINUTE | 5 |
| MINUTE TO SECOND | 13 |
| MONTH | 2 |
| SECOND | 6 |
| YEAR | 1 |
| YEAR TO MONTH | 7 |



### 5.3.9 Other tables associated with CLI

**Table 10—SQL-statement integer codes for use in a diagnostics area**

| SQL-statement | Code |
|---|---|
| <alter domain statement> | 3 |
| <alter table statement> | 4 |
| <assertion definition> | 6 |

**Table 10—SQL-statement integer codes for use in a diagnostics area (Cont.)**

| SQL-statement | Code |
|---|---|
| <call statement>[1] | 7 |
| <character set definition> | 8 |
| <collation definition> | 10 |
| <delete statement: searched> | 19 |
| <domain definition> | 23 |
| <drop assertion statement> | 24 |
| <drop character set statement> | 25 |
| <drop collation statement> | 26 |
| <drop domain statement> | 27 |
| <drop schema statement | 31 |
| <drop table statement> | 32 |
| <drop translation statement> | 33 |
| <drop view statement> | 36 |
| <dynamic select statement> | 85 |
| <dynamic single row select statement> | 41 |
| <grant statement> | 48 |
| <insert statement> | 50 |
| <preparable dynamic delete statement: positioned> | 54 |
| <preparable dynamic update statement: positioned> | 55 |
| <revoke statement> | 59 |
| <schema definition> | 64 |
| <set catalog statement> | 66 |
| <set constraints mode statement> | 68 |
| <set local time zone statement> | 71 |
| <set names statement> | 72 |
| <set schema statement> | 74 |
| <set transaction statement> | 75 |
| <set session authorization identifier statement> | 76 |
| <table definition> | 77 |
| <translation definition> | 79 |
| <update statement: searched> | 82 |
| <view definition> | 84 |

[1]This statement is defined in Part 4 of ISO/IEC 9075

**Table 11—SQL-statement character codes for use in a diagnostics area**

| SQL-statement | Code |
|---|---|
| <alter domain statement> | ALTER DOMAIN |
| <alter table statement> | ALTER TABLE |
| <assertion definition> | CREATE ASSERTION |
| <call statement>[1] | CALL |
| <character set definition> | CREATE CHARACTER SET |
| <collation definition> | CREATE COLLATION |
| <delete statement: searched> | DELETE WHERE |
| <domain definition> | CREATE DOMAIN |
| <drop assertion statement> | DROP ASSERTION |
| <drop character set statement> | DROP CHARACTER SET |
| <drop collation statement> | DROP COLLATION |
| <drop domain statement> | DROP DOMAIN |
| <drop schema statement | DROP SCHEMA |
| <drop table statement> | DROP TABLE |
| <drop translation statement> | DROP TRANSLATION |
| <drop view statement> | DROP VIEW |
| <dynamic select statement> | SELECT CURSOR |
| <dynamic single row select statement> | SELECT |
| <grant statement> | GRANT |
| <insert statement> | INSERT |
| <preparable dynamic delete statement: positioned> | DYNAMIC DELETE CURSOR |
| <preparable dynamic update statement: positioned> | DYNAMIC UPDATE CURSOR |
| <revoke statement> | REVOKE |
| <schema definition> | CREATE SCHEMA |
| <set catalog statement> | SET CATALOG |
| <set constraints mode statement> | SET CONSTRAINT |
| <set local time zone statement> | SET TIME ZONE |
| <set names statement> | SET NAMES |
| <set schema statement> | SET SCHEMA |
| <set transaction statement> | SET TRANSACTION |
| <set session authorization identifier statement> | SET SESSION AUTHORIZATION |
| <table definition> | CREATE TABLE |
| <translation definition> | CREATE TRANSLATION |

[1]This statement is defined in Part 4 of ISO/IEC 9075

**Table 11—SQL-statement character codes for use in a diagnostics area (Cont.)**

| SQL-statement | Code |
|---|---|
| <update statement: searched> | UPDATE WHERE |
| <view definition> | CREATE VIEW |

**Table 12—Codes used for diagnostic fields**

| Field | Code |
|---|---|
| **Header fields** | |
| DYNAMIC_FUNCTION | 7 |
| DYNAMIC_FUNCTION_CODE | 12 |
| MORE | 13 |
| NUMBER | 2 |
| RETURNCODE | 1 |
| ROW_COUNT | 3 |
| **Fields in status records** | |
| CATALOG_NAME | 18 |
| CLASS_ORIGIN | 8 |
| COLUMN_NAME | 21 |
| CONDITION_NUMBER | 14 |
| CONNECTION_NAME | 10 |
| CONSTRAINT_CATALOG | 15 |
| CONSTRAINT_NAME | 17 |
| CONSTRAINT_SCHEMA | 16 |
| CURSOR_NAME | 22 |
| MESSAGE_LENGTH | 23 |
| MESSAGE_OCTET_LENGTH | 24 |
| MESSAGE_TEXT | 6 |
| NATIVE_CODE | 5 |
| SCHEMA_NAME | 19 |
| SERVER_NAME | 11 |
| SQLSTATE | 4 |
| SUBCLASS_ORIGIN | 9 |
| TABLE_NAME | 20 |

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

**Table 13—Codes used for handle types**

| Handle type | Code |
|---|---|
| CONNECTION HANDLE | 2 |
| DESCRIPTOR HANDLE | 4 |
| ENVIRONMENT HANDLE | 1 |
| STATEMENT HANDLE | 3 |

**Table 14—Codes used for transaction termination**

| Termination type | Code |
|---|---|
| COMMIT | 0 |
| ROLLBACK | 1 |

**Table 15—Codes used for environment attributes**

| Attribute | Code | May be set |
|---|---|---|
| NULL TERMINATION | 10001 | Yes |

**Table 16—Codes used for connection attributes**

| Attribute | Code | May be set |
|---|---|---|
| POPULATE IPD | 10001 | No |

**Table 17—Codes used for statement attributes**

| Attribute | Code | May be set |
|---|---|---|
| APD HANDLE | 10011 | Yes |
| ARD HANDLE | 10010 | Yes |
| IPD HANDLE | 10013 | No |
| IRD HANDLE | 10012 | No |
| CURSOR SCROLLABLE | −1 | Yes |
| CURSOR SENSITIVITY | −2 | Yes |

**Table 18—Codes used for FreeStmt options**

| Option | Code |
|---|---|
| CLOSE CURSOR | 0 |
| FREE HANDLE | 1 |
| UNBIND COLUMNS | 2 |
| UNBIND PARAMETERS | 3 |

**Table 19—Data types of attributes**

| Attribute | Data type | Values |
|---|---|---|
| **Environment attributes** | | |
| NULL TERMINATION | INTEGER | 0 ( *false* ) ; 1 ( *true* ) |
| **Connection attributes** | | |
| POPULATE IPD | INTEGER | 0 ( *false* ) ; 1 ( *true* ) |
| **Statement attributes** | | |
| APD HANDLE | INTEGER | Handle value |
| ARD HANDLE | INTEGER | Handle value |
| IPD HANDLE | INTEGER | Handle value |
| IRD HANDLE | INTEGER | Handle value |
| CURSOR SCROLLABLE | INTEGER | 0 (NONSCROLLABLE)<br>1 (SCROLLABLE) |
| CURSOR SENSITIVITY | INTEGER | 0 (UNSPECIFIED)<br>1 (INSENSITIVE) |

**Table 20—Codes used for descriptor fields**

| Field | Code | SQL Item Descriptor Name |
|---|---|---|
| ALLOC_TYPE | 1099 | (*Not applicable*) |
| CHARACTER_SET_CATALOG | 1018 | CHARACTER_SET_CATALOG |
| CHARACTER_SET_NAME | 1020 | CHARACTER_SET_NAME |
| CHARACTER_SET_SCHEMA | 1019 | CHARACTER_SET_SCHEMA |
| COLLATION_CATALOG | 1015 | COLLATION_CATALOG |
| COLLATION_NAME | 1017 | COLLATION_NAME |
| COLLATION_SCHEMA | 1016 | COLLATION_SCHEMA |
| COUNT | 1001 | COUNT |
| DATA_POINTER | 1010 | DATA |
| DATETIME_INTERVAL_CODE | 1007 | DATETIME_INTERVAL_CODE |
| DATETIME_INTERVAL_PRECISION | 1014 | DATETIME_INTERVAL_PRECISION |
| INDICATOR_POINTER | 1009 | INDICATOR |
| LENGTH | 1003 | LENGTH |
| NAME | 1011 | NAME |
| NULLABLE | 1008 | NULLABLE |
| OCTET_LENGTH | 1013 | OCTET_LENGTH |
| OCTET_LENGTH_POINTER | 1004 | Both OCTET_LENGTH (input) and RETURNED_OCTET_LENGTH (output) |
| PARAMETER_MODE | 1021 | PARAMETER_MODE[1] |
| PARAMETER_ORDINAL_POSITION | 1022 | PARAMETER_ORDINAL_POSITION[1] |
| PARAMETER_SPECIFIC_CATALOG | 1023 | PARAMETER_SPECIFIC_CATALOG[1] |
| PARAMETER_SPECIFIC_NAME | 1024 | PARAMETER_SPECIFIC_NAME[1] |
| PARAMETER_SPECIFIC_SCHEMA | 1025 | PARAMETER_SPECIFIC_SCHEMA[1] |
| PRECISION | 1005 | PRECISION |
| SCALE | 1006 | SCALE |
| TYPE | 1002 | TYPE |
| UNNAMED | 1012 | UNNAMED |

[1]These fields are defined in Part 4 of ISO/IEC 9075

**Table 21—Codes used for fetch orientation**

| Fetch Orientation | Code |
|---|---|
| NEXT | 1 |
| FIRST | 2 |
| LAST | 3 |
| PRIOR | 4 |
| ABSOLUTE | 5 |
| RELATIVE | 6 |

**Table 22—Miscellaneous codes used in CLI**

| Context | Code | Indicates |
|---|---|---|
| Allocation type | 1 | AUTOMATIC |
| Allocation type | 2 | USER |
| Attribute value | 0 | FALSE, NONSCROLLABLE, UNSPECIFIED |
| Attribute value | 1 | TRUE, SCROLLABLE, INSENSITIVE |
| Data type | 0 | ALL TYPES |
| Data type | −99 | ARD TYPE |
| Data type | 99 | DEFAULT |
| Input string length | −3 | NULL TERMINATED |
| Parameter length | −2 | DATA AT EXEC |

**Table 23—Codes used for GetData data types**

| Data Type | Code |
|---|---|
| CHARACTER | 1 |
| DOUBLE PRECISION | 8 |
| INTEGER | 4 |
| REAL | 7 |
| SMALLINT | 5 |

**Table 24—Codes used to identify SQL/CLI routines**

| Generic Name | Code |
|---|---|
| AllocConnect | 1 |
| AllocEnv | 2 |
| AllocHandle | 1001 |
| AllocStmt | 3 |

Table 24—Codes used to identify SQL/CLI routines (Cont.)

| Generic Name | Code |
|---|---|
| BindCol | 4 |
| BindParam | 1002 |
| Cancel | 5 |
| CloseCursor | 1003 |
| ColAttribute | 6 |
| Connect | 7 |
| CopyDesc | 1004 |
| DataSources | 57 |
| DescribeCol | 8 |
| Disconnect | 9 |
| EndTran | 1005 |
| Error | 10 |
| ExecDirect | 11 |
| Execute | 12 |
| Fetch | 13 |
| FetchScroll | 1021 |
| FreeConnect | 14 |
| FreeEnv | 15 |
| FreeHandle | 1006 |
| FreeStmt | 16 |
| GetConnectAttr | 1007 |
| GetCursorName | 17 |
| GetData | 43 |
| GetDescField | 1008 |
| GetDescRec | 1009 |
| GetDiagField | 1010 |
| GetDiagRec | 1011 |
| GetEnvAttr | 1012 |
| GetFunctions | 44 |
| GetInfo | 45 |
| GetStmtAttr | 1014 |
| GetTypeInfo | 47 |
| NumResultCols | 18 |
| ParamData | 48 |

**Table 24—Codes used to identify SQL/CLI routines (Cont.)**

| Generic Name | Code |
|---|---|
| Prepare | 19 |
| PutData | 49 |
| RowCount | 20 |
| SetConnectAttr | 1016 |
| SetCursorName | 21 |
| SetDescField | 1017 |
| SetDescRec | 1018 |
| SetEnvAttr | 1019 |
| SetStmtAttr | 1020 |

**Table 25—Codes and data types for implementation information**

| Information Type | Code | Data Type |
|---|---|---|
| ALTER TABLE | 86 | INTEGER |
| CATALOG NAME | 10003 | CHARACTER(1) |
| COLLATING SEQUENCE | 10004 | CHARACTER(254) |
| CURSOR COMMIT BEHAVIOR | 23 | SMALLINT |
| CURSOR SENSITIVITY | 10001 | INTEGER |
| DATA SOURCE NAME | 2 | CHARACTER(128) |
| DATA SOURCE READ ONLY | 25 | CHARACTER(1) |
| DBMS NAME | 17 | CHARACTER(254) |
| DBMS VERSION | 18 | CHARACTER(254) |
| DEFAULT TRANSACTION ISOLATION | 26 | INTEGER |
| DESCRIBE PARAMETER | 10002 | CHARACTER(1) |
| FETCH DIRECTION | 8 | INTEGER |
| GETDATA EXTENSIONS | 81 | INTEGER |
| IDENTIFIER CASE | 28 | SMALLINT |
| INTEGRITY | 73 | CHARACTER(1) |
| MAXIMUM CATALOG NAME LENGTH | 34 | SMALLINT |
| MAXIMUM COLUMN NAME LENGTH | 30 | SMALLINT |
| MAXIMUM COLUMNS IN GROUP BY | 97 | SMALLINT |
| MAXIMUM COLUMNS IN ORDER BY | 99 | SMALLINT |
| MAXIMUM COLUMNS IN SELECT | 100 | SMALLINT |
| MAXIMUM COLUMNS IN TABLE | 101 | SMALLINT |
| MAXIMUM CONCURRENT ACTIVITIES | 1 | SMALLINT |

**Table 25—Codes and data types for implementation information (Cont.)**

| Information Type | Code | Data Type |
|---|---|---|
| MAXIMUM CURSOR NAME LENGTH | 31 | SMALLINT |
| MAXIMUM DRIVER CONNECTIONS | 0 | SMALLINT |
| MAXIMUM IDENTIFIER LENGTH | 10005 | SMALLINT |
| MAXIMUM SCHEMA NAME LENGTH | 32 | SMALLINT |
| MAXIMUM STATEMENT LENGTH | 105 | SMALLINT |
| MAXIMUM TABLE NAME LENGTH | 35 | SMALLINT |
| MAXIMUM TABLES IN SELECT | 106 | SMALLINT |
| MAXIMUM USER NAME LENGTH | 107 | SMALLINT |
| NULL COLLATION | 85 | SMALLINT |
| OUTER JOIN CAPABILITIES | 115 | CHARACTER(1) |
| ORDER BY COLUMNS IN SELECT | 90 | CHARACTER(1) |
| SCROLL CONCURRENCY | 43 | INTEGER |
| SERVER NAME | 13 | CHARACTER(128) |
| SPECIAL CHARACTERS | 94 | CHARACTER(254) |
| TRANSACTION CAPABLE | 46 | SMALLINT |
| TRANSACTION ISOLATION OPTION | 72 | INTEGER |
| USER NAME | 47 | CHARACTER(128) |
| Implementation-defined information type | Implementation-defined | Implementation-defined |

**Table 26—Values for ALTER TABLE with GetInfo**

| Information Type | Value |
|---|---|
| ADD COLUMN | 1 |
| DROP COLUMN | 2 |
| ALTER COLUMN | 4 |
| ADD CONSTRAINT | 8 |
| DROP CONSTRAINT | 16 |

**Table 27—Values for CURSOR COMMIT BEHAVIOR with GetInfo**

| Information Type | Value |
|---|---|
| DELETE | 0 |
| CLOSE | 1 |
| PRESERVE | 2 |

Table 28—Values for FETCH DIRECTION with GetInfo

| Information Type | Value |
|---|---|
| FETCH NEXT | 1 |
| FETCH FIRST | 2 |
| FETCH LAST | 4 |
| FETCH PRIOR | 8 |
| FETCH ABSOLUTE | 16 |
| FETCH RELATIVE | 32 |

Table 29—Values for GETDATA EXTENSIONS with GetInfo

| Information Type | Value |
|---|---|
| ANY COLUMN | 1 |
| ANY ORDER | 2 |

Table 30—Values for IDENTIFIER CASE with GetInfo

| Information Type | Value |
|---|---|
| UPPER | 1 |
| LOWER | 2 |
| SENSITIVE | 3 |
| MIXED | 4 |

Table 31—Values for OUTER JOIN CAPABILITIES with GetInfo

| Information Type | Value |
|---|---|
| LEFT | 1 |
| RIGHT | 2 |
| FULL | 4 |
| NESTED | 8 |
| NOT ORDERED | 16 |
| INNER | 32 |
| ALL COMPARISON OPS | 64 |

Table 32—Values for SCROLL CONCURRENCY with GetInfo

| Information Type | Value |
|---|---|
| READ ONLY | 1 |
| LOCK | 2 |
| OPT ROWVER | 4 |
| OPT VALUES | 8 |

Table 33—Values for TRANSACTION CAPABLE with GetInfo

| Information Type | Value |
|---|---|
| NONE | 0 |
| DML | 1 |
| ALL | 2 |
| DDL COMMIT | 3 |
| DDL IGNORE | 4 |

Table 34—Values for TRANSACTION ISOLATION OPTION with GetInfo

| Information Type | Value |
|---|---|
| READ UNCOMMITTED | 1 |
| READ COMMITTED | 2 |
| REPEATABLE READ | 4 |
| SERIALIZABLE | 8 |

Table 35—Values for NULL COLLATION with GetInfo

| Information Type | Value |
|---|---|
| HIGH | 0 |
| LOW | 1 |

**Table 36—Codes used for concise data types**

| Data Type | Code |
|---|---|
| Implementation-defined data type | <0 |
| CHARACTER | 1 |
| CHAR | 1 |
| NUMERIC | 2 |
| DECIMAL | 3 |
| DEC | 3 |
| INTEGER | 4 |
| INT | 4 |
| SMALLINT | 5 |
| FLOAT | 6 |
| REAL | 7 |
| DOUBLE | 8 |
| CHARACTER VARYING | 12 |
| CHAR VARYING | 12 |
| VARCHAR | 12 |
| BIT | 14 |
| BIT VARYING | 15 |
| DATE | 91 |
| TIME | 92 |
| TIMESTAMP | 93 |
| TIME WITH TIME ZONE | 94 |
| TIMESTAMP WITH TIME ZONE | 95 |
| INTERVAL YEAR | 101 |
| INTERVAL MONTH | 102 |
| INTERVAL DAY | 103 |
| INTERVAL HOUR | 104 |
| INTERVAL MINUTE | 105 |
| INTERVAL SECOND | 106 |
| INTERVAL YEAR TO MONTH | 107 |
| INTERVAL DAY TO HOUR | 108 |

Table 36—Codes used for concise data types (Cont.)

| Data Type | Code |
|---|---|
| INTERVAL DAY TO MINUTE | 109 |
| INTERVAL DAY TO SECOND | 110 |
| INTERVAL HOUR TO MINUTE | 111 |
| INTERVAL HOUR TO SECOND | 112 |
| INTERVAL MINUTE TO SECOND | 113 |

Table 37—Codes used with concise datetime data types in SQL/CLI

| Concise Data Type Code | Data Type Code | Datetime Interval Code |
|---|---|---|
| 91 | 9 | 1 |
| 92 | 9 | 2 |
| 93 | 9 | 3 |
| 94 | 9 | 4 |
| 95 | 9 | 5 |

Table 38—Codes used with concise interval data types in SQL/CLI

| Concise Data Type Code | Data Type Code | Datetime Interval Code |
|---|---|---|
| 101 | 10 | 1 |
| 102 | 10 | 2 |
| 103 | 10 | 3 |
| 104 | 10 | 4 |
| 105 | 10 | 5 |
| 106 | 10 | 6 |
| 107 | 10 | 7 |
| 108 | 10 | 8 |
| 109 | 10 | 9 |
| 110 | 10 | 10 |
| 111 | 10 | 11 |
| 112 | 10 | 12 |
| 113 | 10 | 13 |

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

| Datetime Interval Code | Code |
|---|---|
| 1 | 91 |
| 2 | 92 |
| 3 | 93 |
| 4 | 94 |
| 5 | 95 |

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

| Datetime Interval Code | Code |
|---|---|
| 1 | 101 |
| 2 | 102 |
| 3 | 103 |
| 4 | 104 |
| 5 | 105 |
| 6 | 106 |
| 7 | 107 |
| 8 | 108 |
| 9 | 109 |
| 10 | 110 |
| 11 | 111 |
| 12 | 112 |
| 13 | 113 |

# 5.4 Data type correspondences

## Function

Specify the data type correspondences for SQL data types and host language types associated with the required parameter mechanisms, as shown in Table 2, "Supported calling conventions of CLI routines by language".

In the following tables, let *P* be <precision>, *S* be <scale>, *L* be <length>, *T* be <time fractional seconds precision>, and *Q* be <interval qualifier>.

## Tables

**Table 41—Data type correspondences for Ada**

| SQL Data Type | Ada Data Type |
|---|---|
| CHARACTER (*L*) | SQL_STANDARD.CHAR, with P'LENGTH of *L* |
| CHARACTER VARYING (*L*) | *None* |
| BIT (*L*) | SQL_STANDARD.BIT, with P'LENGTH of *L* |
| BIT VARYING (*L*) | *None* |
| SMALLINT | SQL_STANDARD.SMALLINT |
| INTEGER | SQL_STANDARD.INT |
| DECIMAL(*P*,*S*) | *None* |
| NUMERIC(*P*,*S*) | *None* |
| REAL | SQL_STANDARD.REAL |
| DOUBLE PRECISION | SQL_STANDARD.DOUBLE_PRECISION |
| FLOAT(*P*) | *None* |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

**Table 42—Data type correspondences for C**

| SQL Data Type | C Data Type |
|---|---|
| CHARACTER (*L*) | char, with length *L*+*k*[2] |
| CHARACTER VARYING (*L*) | char, with length *L*+*k*[2] |
| BIT (*L*) | char, with length *X*[1] |
| BIT VARYING (*L*) | *None* |
| SMALLINT | short |
| INTEGER | long |
| DECIMAL(*P*,*S*) | *None* |
| NUMERIC(*P*,*S*) | *None* |
| REAL | float |
| DOUBLE PRECISION | double |
| FLOAT(*P*) | *None* |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a character of the host language.

[2]*k* is the length in units of C **char** of the largest character in the character set associated with the SQL data type.

## 5.4 Data type correspondences

**Table 43—Data type correspondences for COBOL**

| SQL Data Type | COBOL Data Type |
|---|---|
| CHARACTER (*L*) | alphanumeric, with length *L* |
| CHARACTER VARYING (*L*) | *None* |
| BIT (*L*) | alphanumeric, with length *X*[1] |
| BIT VARYING (*L*) | *None* |
| SMALLINT | PICTURE S9(*SPI*) USAGE BINARY, where *SPI* is implementation-defined |
| INTEGER | PICTURE S9(*PI*) USAGE BINARY, where *PI* is implementation-defined |
| DECIMAL(*P*,*S*) | *None* |
| NUMERIC(*P*,*S*) | USAGE DISPLAY SIGN LEADING SEPARATE, with PICTURE as specified[2] |
| REAL | *None* |
| DOUBLE PRECISION | *None* |
| FLOAT(*P*) | *None* |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

[1]The length of a character type corresponding with SQL BIT(*L*) is one more than the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in one character of the host language.

[2]Case:

a) If *S*=*P*, then a PICTURE with an 'S' followed by a 'V' followed by *P* '9's.
b) If *P*>*S*>0, then a PICTURE with an 'S' followed by *P*–*S* '9's followed by a 'V' followed by *S* '9's.
c) If *S*=0, then a PICTURE with an 'S' followed by *P* '9's optionally followed by a 'V'.

**Table 44—Data type correspondences for Fortran**

| SQL Data Type | Fortran Data Type |
|---|---|
| CHARACTER (*L*) | CHARACTER, with length *L* |
| CHARACTER VARYING (*L*) | *None* |
| BIT (*L*) | CHARACTER, with length *X*[1] |
| BIT VARYING (*L*) | *None* |
| SMALLINT | *None* |
| INTEGER | INTEGER |
| DECIMAL(*P*,*S*) | *None* |
| NUMERIC(*P*,*S*) | *None* |
| REAL | REAL |
| DOUBLE PRECISION | DOUBLE PRECISION |
| FLOAT(*P*) | *None* |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a character of the host language.

**Table 45—Data type correspondences for MUMPS**

| SQL Data Type | MUMPS Data Type |
|---|---|
| CHARACTER (*L*) | *None* |
| CHARACTER VARYING (*L*) | character with maximum length *L* |
| BIT (*L*) | *None* |
| BIT VARYING (*L*) | *None* |
| SMALLINT | *None* |
| INTEGER | character |
| DECIMAL(*P*,*S*) | character |
| NUMERIC(*P*,*S*) | character |
| REAL | character |
| DOUBLE PRECISION | *None* |
| FLOAT(*P*) | *None* |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

**Table 46—Data type correspondences for Pascal**

| SQL Data Type | Pascal Data Type |
|---|---|
| CHARACTER(1) | CHAR |
| CHARACTER (*L*), *L* > 1 | PACKED ARRAY[1..*L*] OF CHAR |
| CHARACTER VARYING (*L*) | *None* |
| BIT (*L*), 1 ≤ *L* ≤ *B*[1] | CHAR |
| BIT (*L*), *B*[1] < *L* | PACKED ARRAY[LB[1]] OF CHAR |
| BIT VARYING (*L*) | *None* |
| SMALLINT | *None* |
| INTEGER | INTEGER |
| DECIMAL(*P*,*S*) | *None* |
| NUMERIC(*P*,*S*) | *None* |
| REAL | REAL |
| DOUBLE PRECISION | *None* |
| FLOAT(*P*) | *None* |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

[1]The length *LB* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a character of the host language.

**Table 47—Data type correspondences for PL/I**

| SQL Data Type | PL/I Data Type |
|---|---|
| CHARACTER (*L*) | CHARACTER(*L*) |
| CHARACTER VARYING (*L*) | CHARACTER VARYING(*L*) |
| BIT (*L*) | BIT(*L*) |
| BIT VARYING (*L*) | BIT VARYING (*L*) |
| SMALLINT | FIXED BINARY(*SPI*), where *SPI* is implementation-defined |
| INTEGER | FIXED BINARY(*PI*), where *PI* is implementation-defined |
| DECIMAL(*P*,*S*) | FIXED DECIMAL(*P*,*S*) |
| NUMERIC(*P*,*S*) | *None* |
| REAL | *None* |
| DOUBLE PRECISION | *None* |
| FLOAT(*P*) | FLOAT BINARY (*P*) |
| DATE | *None* |
| TIME(*T*) | *None* |
| TIMESTAMP(*T*) | *None* |
| INTERVAL(*Q*) | *None* |

# 6 SQL/CLI routines

Subclause 5.1, "<CLI routine>", defines a generic CLI routine. This Subclause describes the individual CLI routines in alphabetical order.

For convenience, the variable <CLI name prefix> is omitted and the <CLI generic name> is used for the descriptions. For presentation purposes (and purely arbitrarily), the routines are presented as functions rather than as procedures.

## 6.1 AllocConnect

### Function

Allocate an SQL-connection and assign a handle to it.

### Definition

```
AllocConnect (
      EnvironmentHandle      IN        INTEGER,
      ConnectionHandle       OUT       INTEGER )
      RETURNS SMALLINT
```

### General Rules

1) Let *EH* be the value of EnvironmentHandle.

2) AllocHandle is implicitly invoked with HandleType indicating CONNECTION HANDLE, with *EH* as the value of InputHandle and with ConnectionHandle as OutputHandle.

## 6.2 AllocEnv

### Function

Allocate an SQL-environment and assign a handle to it.

### Definition

```
AllocEnv (
      EnvironmentHandle      OUT      INTEGER )
      RETURNS SMALLINT
```

### General Rules

1) AllocHandle is implicitly invoked with HandleType indicating ENVIRONMENT HANDLE, with zero as the value of InputHandle, and with EnvironmentHandle as OutputHandle.

# 6.3 AllocHandle

## Function

Allocate a resource and assign a handle to it.

## Definition

```
AllocHandle (
      HandleType        IN        SMALLINT,
      InputHandle       IN        INTEGER,
      OutputHandle      OUT       INTEGER )
      RETURNS SMALLINT
```

## General Rules

1) Let *HT* be the value of HandleType and let *IH* be the value of InputHandle.

2) If *HT* is not one of the code values in Table 13, "Codes used for handle types", then an exception condition is raised: *CLI-specific condition — invalid handle*.

3) Case:

   a) If *HT* indicates ENVIRONMENT HANDLE, then:

      i) If the maximum number of SQL-environments that can be allocated at one time has already been reached, then an exception condition is raised: *CLI-specific condition — limit on number of handles exceeded*. A skeleton SQL-environment is allocated and is assigned a unique value that is returned in OutputHandle.

      ii) Case:

         1) If the memory requirements to manage an SQL-environment cannot be satisfied, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — memory allocation error*.

            NOTE 9 – No diagnostic information is generated in this case as there is no valid environment handle that can be used in order to obtain diagnostic information.

         2) If the resources to manage an SQL-environment cannot be allocated for implementation-defined reasons, then an implementation-defined exception condition is raised. A skeleton SQL-environment is allocated and is assigned a unique value that is returned in OutputHandle.

         3) Otherwise, the resources to manage an SQL-environment are allocated and are referred to as an allocated SQL-environment. The allocated SQL-environment is assigned a unique value that is returned in OutputHandle.

   b) If *HT* indicates CONNECTION HANDLE, then:

      i) If *IH* does not identify an allocated SQL-environment or if it identifies an allocated skeleton SQL-environment, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Let *E* be the allocated SQL-environment identified by *IH*.

iii) The diagnostics area associated with *E* is emptied.

iv) If the maximum number of SQL-connections that can be allocated at one time has already been reached, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — limit on number of handles exceeded*.

v) Case:

1) If the memory requirements to manage an SQL-connection cannot be satisfied, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — memory allocation error*.

2) If the resources to manage an SQL-connection cannot be allocated for implementation-defined reasons, then OutputHandle is set to zero and an implementation-defined exception condition is raised.

3) Otherwise, the resources to manage an SQL-connection are allocated and are referred to as an allocated SQL-connection. The allocated SQL-connection is associated with *E* and is assigned a unique value that is returned in OutputHandle.

c) If *HT* indicates STATEMENT HANDLE, then:

i) If *IH* does not identify an allocated SQL-connection, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — invalid handle*.

ii) Let *C* be the allocated SQL-connection identified by *IH*.

iii) The diagnostics area associated with *C* is emptied.

iv) If there is no established SQL-connection associated with *C*, then OutputHandle is set to zero and an exception condition is raised: *connection exception — connection does not exist*. Otherwise, let *EC* be the established SQL-connection associated with *C*.

v) If the maximum number of SQL-statements that can be allocated at one time has already been reached, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — limit on number of handles exceeded*.

vi) If *EC* is not the current SQL-connection, then the General Rules of Subclause 5.3.1, "Implicit set connection", are applied to *EC* as the dormant SQL-connection.

vii) Case:

1) If the memory requirements to manage an SQL-statement cannot be satisfied, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — memory allocation error*.

2) If the resources to manage an SQL-statement cannot be allocated for implementation-defined reasons, then OutputHandle is set to zero and an implementation-defined exception condition is raised.

3) Otherwise, the resources to manage an SQL-statement are allocated and are referred to as an allocated SQL-statement. The allocated SQL-statement is associated with *C* and is assigned a unique value that is returned in OutputHandle. The following CLI descriptor areas are automatically allocated and associated with the allocated SQL-statement:

A) An implementation parameter descriptor

B) An implementation row descriptor

C) An application parameter descriptor

D) An application row descriptor

For each of these descriptor areas, the ALLOC_TYPE field is set to indicate AUTOMATIC. The automatically allocated application parameter descriptor becomes the current application parameter descriptor for the allocated SQL-statement and the automatically allocated application row descriptor becomes the current application row descriptor for the allocated SQL-statement.

d) If *HT* indicates DESCRIPTOR HANDLE, then:

i) If *IH* does not identify an allocated SQL-connection then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — invalid handle*.

ii) Let *C* be the allocated SQL-connection identified by *IH*.

iii) The diagnostics area associated with *C* is emptied.

iv) If there is no established SQL-connection associated with *C*, then OutputHandle is set to zero and an exception condition is raised: *connection exception — connection does not exist*. Otherwise, let *EC* be the established SQL-connection associated with *C*.

v) If the maximum number of CLI descriptor areas that can be allocated at one time has already been reached, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — limit on number of handles exceeded*.

vi) If *EC* is not the current SQL-connection, then the General Rules of Subclause 5.3.1, "Implicit set connection", are applied to *EC* as the dormant SQL-connection.

vii) Case:

1) If the memory requirements to manage a CLI descriptor area cannot be satisfied, then OutputHandle is set to zero and an exception condition is raised: *CLI-specific condition — memory allocation error*.

2) If the resources to manage a CLI descriptor area cannot be allocated for implementation-defined reasons, then OutputHandle is set to zero and an implementation-defined exception condition is raised.

3) Otherwise, the resources to manage a CLI descriptor area are allocated and are referred to as an allocated CLI descriptor area. The allocated CLI descriptor area is associated with *C* and is assigned a unique value that is returned in OutputHandle. The ALLOC_TYPE field of the allocated CLI descriptor area is set to indicate USER.

# 6.4 AllocStmt

## Function

Allocate an SQL-statement and assign a handle to it.

## Definition

```
AllocStmt (
       ConnectionHandle       IN       INTEGER )
       StatementHandle        OUT      INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *CH* be the value of ConnectionHandle.

2) AllocHandle is implicitly invoked with HandleType indicating STATEMENT HANDLE, with *CH* as the value of InputHandle, and with StatementHandle as OutputHandle.

# 6.5 BindCol

## Function

Describe a target specification.

## Definition

```
BindCol (
       StatementHandle      IN        INTEGER,
       ColumnNumber         IN        SMALLINT,
       TargetType           IN        SMALLINT,
       TargetValue          DEFOUT    ANY,
       BufferLength         IN        INTEGER,
       StrLen_or_Ind        DEFOUT    INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Let *HV* be the value of the handle of the current application row descriptor for *S*.

3) Case:

   a) If *HV* does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — descriptor invalid on indirect reference*.

   b) Otherwise, let *ARD* be the allocated CLI descriptor area identified by *HV* and let *N* be the value of COUNT for *ARD*.

4) Let *CN* be the value of ColumnNumber.

5) If *CN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

6) Let *TT* be the value of TargetType.

7) 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 this table as the *SQL data type column* and the *host data type column*.

8) If either of the following is true, then an exception condition is raised: *CLI-specific condition — invalid data type in application descriptor*.

   a) *TT* does not indicate DEFAULT and is not one of the code values in Table 7, "Codes used for application data types in CLI".

   b) *TT* is one of the code values in Table 7, "Codes used for application data types in CLI", but the row that contains the corresponding SQL data type in the SQL data type column of the operative data type correspondence table contains 'None' in the host data type column.

9) Let *BL* be the value of BufferLength.

10) If *BL* is not greater than zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

11) Let *IDA* be the item descriptor area of *ARD* specified by *CN*.

12) The data type of the <target specification> described by *IDA* is set to *TT*.

13) The length in octets of the <target specification> described by *IDA* is set to *BL*.

14) The length in characters or positions of the <target specification> described by *IDA* is set to the maximum number of characters or positions that may be represented by the data type *TT*.

15) The address of the host variable that is to receive a value for the <target specification> described by *IDA* is set to the address of TargetValue. If TargetValue is a null pointer, then the address is set to 0.

16) The address of the <indicator variable> associated with the host variable addressed by the DATA_POINTER field of *IDA* is set to the address of StrLen_or_Ind.

17) The address of the host variable that is to receive the returned length (in characters) of the <target specification> described by *IDA* is set to the address of StrLen_or_Ind.

18) If *CN* is greater than *N*, then the COUNT field of *ARD* is set to *CN*.

19) If an exception condition is raised, then the TYPE, OCTET_LENGTH, LENGTH, DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields of *IDA* are set to implementation-dependent values and the value of COUNT for *ARD* is unchanged.

20) Restrictions on the differences allowed between *ARD* and *IRD* are implementation-defined, except as specified in the General Rules of Subclause 5.3.3, "Implicit using clause", and the General Rules of Subclause 6.27, "GetData".

## 6.6 BindParam

### Function

Describe a dynamic parameter specification and its value.

### Definition

```
BindParam (
        StatementHandle         IN          INTEGER,
        ParameterNumber         IN          SMALLINT,
        ValueType               IN          SMALLINT,
        ParameterType           IN          SMALLINT,
        ColumnSize              IN          INTEGER,
        DecimalDigits           IN          SMALLINT,
        ParameterValue          DEFIN       ANY,
        StrLen_or_Ind           DEFIN       INTEGER )
        RETURNS SMALLINT
```

### General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Let *HV* be the value of the handle of the current application parameter descriptor for *S*.

3) Case:

   a) If *HV* does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — descriptor invalid on indirect reference*.

   b) Otherwise, let *APD* be the allocated CLI descriptor area identified by *HV* and let *N2* be the value of COUNT for *APD*.

4) Let *PN* be the value of ParameterNumber.

5) If *PN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

6) Let *VT* be the value of ValueType.

7) 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*.

8) If any of the following are true, then an exception condition is raised: *CLI-specific condition — invalid data type in application descriptor*.

   a) *VT* does not indicate DEFAULT and is not one of the code values in Table 7, "Codes used for application data types in CLI".

   b) *VT* is one of the code values in Table 7, "Codes used for application data types in CLI", but the row that contains the corresponding SQL data type in the SQL data type column of the operative data type correspondence table contains 'None' in the host data type column.

9) Let *PT* be the value of ParameterType.

10) If *PT* is not one of the code values in Table 36, "Codes used for concise data types", then an exception condition is raised: *CLI-specific condition — invalid data type*.

11) Let *IPD* be the implementation parameter descriptor associated with *S* and let *N1* be the value of COUNT for *IPD*.

12) Let *IDA1* be the item descriptor area of *IPD* specified by *PN*.

13) Let *CS* be the value of ColumnSize and let *DD* be the value of DecimalDigits.

14) Case:

    a) If *PT* is one of the values listed in Table 37, "Codes used with concise datetime data types in SQL/CLI", then:

        i) The data type of the <dynamic parameter specification> described by *IDA1* is set to a code shown in the Data Type Code column of Table 37, "Codes used with concise datetime data types in SQL/CLI", indicating the concise data type code.

        ii) The datetime interval code of the <dynamic parameter specification> described by *IDA1* is set to a code shown in the Datetime Interval Code column in Table 37, "Codes used with concise datetime data types in SQL/CLI", indicating the concise data type code.

        iii) The length (in positions) of the <dynamic parameter specification> described by *IDA1* is set to *CS*.

        iv) Case:

            1) If the datetime interval code of the <dynamic parameter specification> indicates DATE, then the time fractional seconds precision of the <dynamic parameter specification> described by *IDA1* is set to zero.

            2) Otherwise, the time fractional seconds precision of the <dynamic parameter specification> described by *IDA1* is set to *DD*.

    b) If *PT* is one of the values listed in Table 38, "Codes used with concise interval data types in SQL/CLI", then:

        i) The data type of the <dynamic parameter specification> described by *IDA1* is set to a code shown in the Data Type Code column of Table 38, "Codes used with concise interval data types in SQL/CLI", indicating the concise data type code.

        ii) The datetime interval code of the <dynamic parameter specification> described by *IDA1* is set to a code shown in the Datetime Interval Code column in Table 38, "Codes used with concise interval data types in SQL/CLI", indicating the concise data type code. Let *DIC* be that code.

        iii) The length (in positions) of the <dynamic parameter specification> described by *IDA1* is set to *CS*.

        iv) Case:

            1) If *DIC* indicates SECOND, DAY TO SECOND, HOUR TO SECOND, or MINUTE TO SECOND, then the interval fractional seconds precision of the <dynamic parameter specification> described by *IDA1* is set to *DD*. If *DD* is zero, then let *DP* be zero; otherwise, let *DP* be one.

2) Otherwise, the interval fractional seconds precision of the <dynamic parameter specification> described by *IDA1* is set to zero.

v) Case:

1) If *DIC* indicates YEAR TO MONTH, DAY TO HOUR, HOUR TO MINUTE or MINUTE TO SECOND, then let *IL* be 3.

2) If *DIC* indicates DAY TO MINUTE or HOUR TO SECOND, then let *IL* be 6.

3) If *DIC* indicates DAY TO SECOND, then let *IL* be 9.

4) Otherwise, let *IL* be zero.

vi) Case:

1) If *DIC* indicates SECOND, DAY TO SECOND, HOUR TO SECOND, or MINUTE TO SECOND, then the interval leading field precision of the <dynamic parameter specification> described by *IDA1* is set to $CS - IL - DD - DP$.

2) Otherwise, the interval leading field precision of the <dynamic parameter specification> described by *IDA1* is set to $CS - IL$.

c) Otherwise:

i) The data type of the <dynamic parameter specification> described by *IDA1* is set to *PT*.

ii) If *PT* indicates a character string type, then the length (in characters) of the <dynamic parameter specification> described by *IDA1* is set to *CS*.

iii) If *PT* indicates a bit type, then the length (in bits) of the <dynamic parameter specification> described by *IDA1* is set to *CS*.

iv) If *PT* indicates a numeric type, then the precision of the <dynamic parameter specification> described by *IDA1* is set to *CS*.

v) If *PT* indicates a numeric type, then the scale of the <dynamic parameter specification> described by *IDA1* is set to *DD*.

15) Let *IDA2* be the item descriptor area of *APD* specified by *PN*.

16) The data type of the <dynamic parameter specification> described by *IDA2* is set to *VT*.

17) The address of the host variable that is to provide a value for the <dynamic parameter specification> value described by *IDA2* is set to the address of ParameterValue.

18) The address of the <indicator variable> associated with the host variable addressed by the DATA_POINTER field of *IDA2* is set to the address of StrLen_or_Ind.

19) The address of the host variable that is to provide the length (in octets) of the <dynamic parameter specification> value described by *IDA2* is set to the address of StrLen_or_Ind.

20) If *PN* is greater than *N1*, then the COUNT field of *IPD* is set to *PN*. If *PN* is greater than *N2*, then the COUNT field of *APD* is set to *PN*.

21) If an exception condition is raised, then:

a) The TYPE, LENGTH, PRECISION, and SCALE fields of *IDA1* are set to implementation-dependent values and the value of COUNT for *IPD* is unchanged.

b) The TYPE, DATA_POINTER, INDICATOR_POINTER, and OCTET_LENGTH_POINTER fields of *IDA2* are set to implementation-dependent values and the value of COUNT for *APD* is unchanged.

22) Restrictions on the differences allowed between *APD* and *IPD* are implementation-defined, except as specified in the General Rules of Subclause 5.3.3, "Implicit using clause", and the General Rules of Subclause 6.38, "ParamData".

# 6.7 Cancel

## Function

Attempt to cancel execution of a CLI routine.

## Definition

```
Cancel (
       StatementHandle IN    INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Case:

   a) If there is a CLI routine concurrently operating on *S*, then:

      i) Let *RN* be the routine name of the concurrent CLI routine.

      ii) Let *C* be the allocated SQL-connection with which *S* is associated.

      iii) Let *EC* be the established SQL-connection associated with *C* and let *SS* be the SQL-server associated with *EC*.

      iv) *SS* is requested to cancel the execution of *RN*.

      v) If *SS* rejects the cancellation request, then an exception condition is raised: *CLI-specific condition — server declined the cancellation request*.

      vi) If *SS* accepts the cancellation request, then a completion condition is raised: *successful completion*.

      NOTE 10 – Acceptance of the request does not guarantee that the execution of RN will be canceled.

      vii) If *SS* succeeds in canceling the execution of *RN*, then an exception condition is raised for *RN*: *CLI-specific condition — operation canceled*.

      NOTE 11 – Canceling the execution of *RN* does not destroy any diagnostic information already generated by its execution.

   NOTE 12 – The method of passing control between concurrently operating programs is implementation-dependent.

   b) If there is a deferred parameter number associated with *S*, then:

      i) The diagnostics area associated with *S* is emptied.

      ii) The deferred parameter number is removed from association with *S*.

      iii) Any statement source associated with *S* is removed from association with *S*.

   c) Otherwise:

      i) The diagnostics area associated with *S* is emptied.

ii) A completion condition is raised: *successful completion*.

## 6.8 CloseCursor

### Function

Close a cursor.

### Definition

```
CloseCursor (
       StatementHandle      IN    INTEGER )
       RETURNS SMALLINT
```

### General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

3) Case:

   a) If there is no open cursor associated with *S*, then an exception condition is raised: *invalid cursor state*.

   b) Otherwise:

      i) The open cursor associated with *S* is placed in the closed state and its copy of the select source is destroyed.

      ii) Any fetched row associated with *S* is removed from association with *S*.

# 6.9 ColAttribute

## Function

Get a column attribute.

## Definition

```
ColAttribute (
       StatementHandle      IN    INTEGER,
       ColumnNumber         IN    SMALLINT,
       FieldIdentifier      IN    SMALLINT,
       CharacterAttribute   OUT   CHARACTER(L),
       BufferLength         IN    SMALLINT,
       StringLength         OUT   SMALLINT,
       NumericAttribute     OUT   INTEGER )
       RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no prepared or executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

3) Let *IRD* be the implementation row descriptor associated with *S* and let *N* be the value of COUNT for *IRD*.

4) Let *FI* be the value of FieldIdentifier.

5) If *FI* is not one of the code values in Table 20, "Codes used for descriptor fields", then an exception condition is raised: *CLI-specific condition — invalid descriptor field identifier*.

6) Let *CN* be the value of ColumnNumber.

7) If *FI* indicates neither ALLOC_TYPE nor COUNT, then:

   a) If *N* is zero, then an exception condition is raised: *dynamic SQL error — prepared statement is not a cursor specification*.

   b) If *CN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

   c) If *CN* is greater than *N*, then a completion condition is raised: *no data*.

8) Let *IDA* be the item descriptor area of *IRD* specified by *CN*.

9) Let *DT* and *DIC* be the values of TYPE and DATETIME_INTERVAL_CODE, respectively, for *IDA*.

10) Let *DH* be the handle that identifies *IRD*.

11) Case:

a) If *FI* indicates TYPE, then:

i) If *DT* indicates a <datetime type>, then NumericAttribute is set to the concise code value corresponding to the datetime interval code value *DIC* as defined in Table 39, "Concise codes used with datetime data types in SQL/CLI".

ii) If *DT* indicates INTERVAL, then NumericAttribute is set to the concise code value corresponding to the datetime interval code value *DIC* as defined in Table 40, "Concise codes used with interval data types in SQL/CLI".

iii) Otherwise, NumericAttribute is set to *DT*.

b) If *FI* indicates NAME, then let the information be retrieved from IRD by implicitly executing GetDescField as follows:

```
GetDescField(DH, CN, FI, CharacterAttribute, BufferLength, StringLength)
```

c) Otherwise, let the information be retrieved from *IRD* by implicitly executing GetDescField as follows:

```
GetDescField(DH, CN, FI, NumericAttribute, BufferLength, StringLength)
```

# 6.10 Connect

## Function

Establish a connection.

## Definition

```
Connect (
       ConnectionHandle     IN    INTEGER,
       ServerName           IN    CHARACTER(L1),
       NameLength1          IN    SMALLINT,
       UserName             IN    CHARACTER(L2),
       NameLength2          IN    SMALLINT,
       Authentication       IN    CHARACTER(L3),
       NameLength3          IN    SMALLINT )
       RETURNS SMALLINT
```

where:

— *L1* is determined by the value of NameLength1 and has a maximum value of 128.

— *L2* is determined by the value of NameLength2 and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

— *L3* is determined by the value of NameLength3 and has an implementation-defined maximum value.

## General Rules

1) Case:

   a) If ConnectionHandle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *C* be the allocated SQL-connection identified by ConnectionHandle.

      ii) The diagnostics area associated with *C* is emptied.

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 there is an established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection name in use*.

4) Case:

   a) If ServerName is a null pointer, then let *NL1* be zero.

   b) Otherwise, let *NL1* be the value of NameLength1.

5) Case:

   a) If *NL1* is not negative, then let *L1* be *NL1*.

   b) If *NL1* indicates NULL TERMINATED, then let *L1* be the number of octets of ServerName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

6) Case:

   a) If *L1* is zero, then let 'DEFAULT' be the value of *SN*.

   b) If *L1* is greater than 128, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

   c) Otherwise, let *SN* be the first *L1* octets of ServerName.

7) Let *E* be the allocated SQL-environment with which *C* is associated.

8) Case:

   a) If UserName is a null pointer, then let *NL2* be zero.

   b) Otherwise, let *NL2* be the value of NameLength2.

9) Case:

   a) If *NL2* is not negative, then let *L2* be *NL2*.

   b) If *NL2* indicates NULL TERMINATED, then let *L2* be the number of Octets of UserName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

10) Case:

   a) If Authentication is a null pointer, then let *NL3* be zero.

   b) Otherwise, let *NL3* be the value of NameLength3.

11) Case:

   a) If *NL3* is not negative, then let *L3* be *NL3*.

   b) If *NL3* indicates NULL TERMINATED, then let *L3* be the number of octets of Authentication that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

12) Case:

a) If the value of *SN* is 'DEFAULT', then:

i) If *L2* is not zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

ii) If *L3* is not zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

iii) If an established default SQL-connection is associated with an allocated SQL-connection associated with E, then an exception condition is raised: *connection exception — connection name in use*.

b) Otherwise:

i) If *L2* is zero, then let *UN* be an implementation-defined <authorization identifier>.

ii) If *L2* is non-zero, then:

1) Let *UV* be the first *L2* octets of UserName and let *UN* be the result of

```
TRIM ( BOTH ' ' FROM UV )
```

2) If *UN* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid authorization specification*.

NOTE 13 – An <identifier> may optionally contain "<introducer><character set name>". See Subclause 5.4, "Names and identifiers", in ISO/IEC 9075:1992.

3) If *UN* violates any implementation-defined restrictions on its value, then an exception condition is raised: *invalid authorization specification*.

iii) Case:

1) If *L3* is not zero, then let *AU* be the first *L3* octets of Authentication.

2) Otherwise, let *AU* be an implementation-defined authentication string, whose length may be zero.

13) Case:

a) If the value of *SN* is 'DEFAULT', then the default SQL-session is initiated and associated with the default SQL-server. The method by which the default SQL-server is determined is implementation-defined.

b) Otherwise, an SQL-session is initiated and associated with the SQL-server identified by *SN*. The method by which *SN* is used to determine the appropriate SQL-server is implementation-defined.

14) If an SQL-session is successfully initiated, then:

a) The current SQL-connection and current SQL-session, if any, 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 initiated SQL-connection.

NOTE 14 – The SQL-session context information is defined in Subclause 4.30, "SQL-sessions", in ISO/IEC 9075:1992.

b) The initiated SQL-session becomes the current SQL-session and the SQL-connection established to that SQL-session becomes the current SQL-connection and is associated with *C*.

NOTE 15 – If an SQL-session is not successfully initiated, then the current SQL-connection and current SQL-session, if any, remain unchanged.

15) If the SQL-client cannot establish the SQL-connection, then an exception condition is raised: *connection exception — SQL-client unable to establish SQL-connection*.

16) If the SQL-server rejects the establishment of the SQL-connection, then an exception condition is raised: *connection exception — SQL-server rejected establishment of SQL-connection*.

NOTE 16 – *AU* and *UN* are used by the SQL-server, along with other implementation-dependent values, to determined whether to accept or reject the establishment of an SQL-session.

17) The SQL-server for the subsequent execution of SQL-statements via CLI routine invocations is set to the SQL-server identified by *SN*.

18) The SQL-session <authorization identifier> is set to *UN*.

# 6.11 CopyDesc

## Function

Copy a CLI descriptor.

## Definition

```
CopyDesc (
       SourceDescHandle      IN        INTEGER,
       TargetDescHandle      IN        INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Case:

   a) If SourceDescHandle does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise, let *SD* be the CLI descriptor area identified by SourceDescHandle.

2) Case:

   a) If TargetDescHandle does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *TD* be the CLI descriptor area identified by TargetDescHandle.

      ii) The diagnostics area associated with *TD* is emptied.

3) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *SD* as the DESCRIPTOR AREA.

4) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *TD* as the DESCRIPTOR AREA.

5) If *TD* is an implementation row descriptor, then an exception condition is raised: *CLI-specific condition — cannot modify an implementation row descriptor*.

6) Let *AT* be the value of the ALLOC_TYPE field of *TD*.

7) The contents of *TD* are replaced by a copy of the contents of *SD*.

8) The ALLOC_TYPE field of *TD* is set to *AT*.

## 6.12 DataSources

### Function

Get server name(s) that the application can connect to, along with description information, if available.

### Definition

```
DataSources (
        EnvironmentHandle       IN      INTEGER,
        Direction               IN      SMALLINT,
        ServerName              OUT     CHARACTER(L1),
        BufferLength1           IN      SMALLINT,
        NameLength1             OUT     SMALLINT,
        Description             OUT     CHARACTER(L2),
        BufferLength2           IN      SMALLINT,
        NameLength2             OUT     SMALLINT )
        RETURNS SMALLINT
```

where *L1* and *L2* are the values of BufferLength1 and BufferLength2, respectively, and have maximum values equal to the implementation-defined maximum length of a variable-length character string.

### General Rules

1) Let *EH* be the value of EnvironmentHandle.

2) If *EH* does not identify an allocated SQL-environment or if it identifies an allocated skeleton SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

3) Let *E* be the allocated SQL-environment identified by *EH*. The diagnostics area associated with *E* is emptied.

4) Let *BL1* and *BL2* be the values of BufferLength1 and BufferLength2, respectively.

5) Let *D* be the value of Direction.

6) If *D* is not either the code value for NEXT or the code value for FIRST in Table 21, "Codes used for fetch orientation", then an exception condition is raised: *CLI-specific condition — invalid retrieval code*.

7) Let $SN_1$, $SN_2$, $SN_3$, *etc.*, be an ordered set of the names of SQL-servers to which the application might be eligible to connect (where the mechanism used to establish this set is implementation-defined).

   NOTE 17 – $SN_1$, $SN_2$, $SN_3$, *etc.*, are the names that an application program would use in invocations of Connect, rather than the "actual" names of the SQL-servers.

8) Let $D_1$, $D_2$, $D_3$, *etc.*, be strings describing the SQL-servers named by $SN_1$, $SN_2$, $SN_3$, *etc.* (again provided via an implementation-defined mechanism).

9) Case:

a) If *D* indicates FIRST, or if DataSources has never been successfully called on *EH*, or if the previous call to DataSources on *EH* raised a completion condition: *no data*, then:

i) If there are no entries in the set $SN_1$, $SN_2$, $SN_3$, *etc.*, then a completion condition is raised: *no data* and no further rules for this Subclause are applied.

ii) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with ServerName, $SN_1$, *BL1*, and NameLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

iii) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with Description, $D_1$, *BL2*, and NameLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

b) Otherwise,

i) Let $SN_n$ be the ServerName value that was returned on the previous call to DataSources on *EH*.

ii) If there is no entry in the set after $SN_n$, then a completion condition is raised: *no data* and no further rules for this subclause are applied.

iii) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with ServerName, $SN_{n+1}$, *BL1*, and NameLength1 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

iv) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with Description, $D_{n+1}$, *BL2*, and NameLength2 as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

# 6.13 DescribeCol

## Function

Get column attributes.

## Definition

```
DescribeCol (
       StatementHandle       IN        INTEGER,
       ColumnNumber          IN        SMALLINT,
       ColumnName            OUT       CHARACTER(L)
       BufferLength          IN        SMALLINT,
       NameLength            OUT       SMALLINT,
       DataType              OUT       SMALLINT,
       ColumnSize            OUT       INTEGER,
       DecimalDigits         OUT       SMALLINT,
       Nullable              OUT       SMALLINT )
       RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no prepared or executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

3) Let *IRD* be the implementation row descriptor associated with *S* and let *N* be the value of COUNT for *IRD*.

4) If *N* is zero, then an exception condition is raised: *dynamic SQL error — prepared statement not a cursor specification*.

5) Let *CN* be the value of ColumnNumber.

6) If *CN* is less than 1 or greater than *N*, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

7) Let *C* be the <select list> column described by the item descriptor area of *IRD* specified by *CN*.

8) Let *BL* be the value of BufferLength.

9) Information is retrieved from *IRD*:

   a) Case:

      i) If the data type of *C* is datetime, then DataType is set to the value of the Code column from Table 39, "Concise codes used with datetime data types in SQL/CLI", corresponding to the datetime interval code of *C*.

      ii) If the data type of *C* is interval, then DataType is set to the value of the Code column from Table 40, "Concise codes used with interval data types in SQL/CLI", corresponding to the datetime interval code of *C*.

iii) Otherwise, DataType is set to the data type of *C*.

b) Case:

i) If the data type of *C* is character string, then ColumnSize is set to the maximum length in octets of *C*.

ii) If the data type of *C* is exact numeric or approximate numeric, then ColumnSize is set to the maximum length of *C* in decimal digits.

iii) If the data type of *C* is datetime or interval, then ColumnSize is set to the length in positions of *C*.

iv) If the data type of *C* is bit string, then ColumnSize is set to the length in bits of *C*.

c) Case:

i) If the data type of *C* is exact numeric, then DecimalDigits is set to the scale of *C*.

ii) If the data type of *C* is datetime, then DecimalDigits is set to the time fractional seconds precision of *C*.

iii) If the data type of *C* is interval, then DecimalDigits is set to the interval fractional seconds precision of *C*.

iv) Otherwise, DecimalDigits is set to an implementation-dependent value.

d) If *C* can have the null value, then Nullable is set to 1; otherwise, Nullable is set to 0.

e) The name associated with *C* is retrieved. If *C* has an implementation-dependent name, then the value retrieved is the implementation-dependent name for *C*; otherwise, the value retrieved is the <derived column> name of *C*. Let *V* be the value retrieved. The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with ColumnName, *V*, *BL*, and NameLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

# 6.14 Disconnect

## Function

Terminate an established connection.

## Definition

```
Disconnect (
      ConnectionHandle     IN    INTEGER )
      RETURNS SMALLINT
```

## General Rules

1) Case:

   a) If ConnectionHandle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *C* be the allocated SQL-connection identified by ConnectionHandle.

      ii) The diagnostics area associated with *C* is emptied.

2) Case:

   a) If there is no established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection does not exist*.

   b) Otherwise, let *EC* be the established SQL-connection associated with *C*.

3) Let *L1* be a list of the allocated SQL-statements associated with *C*. Let *L2* be a list of the allocated CLI descriptor areas associated with *C*.

4) If *EC* is active, then

   Case:

   a) If any allocated SQL-statement in *L1* has a deferred parameter number associated with it, then an exception condition is raised: *CLI-specific condition — function sequence error*.

   b) Otherwise, an exception condition is raised: *invalid transaction state*.

5) For every allocated SQL-statement *AS* in *L1*:

   a) Let *SH* be the StatementHandle that identifies *AS*.

   b) FreeHandle is implicitly invoked with HandleType indicating STATEMENT HANDLE and with *SH* as the value of Handle.

   NOTE 18 – Any diagnostic information generated by the invocation is associated with *C* and not with *AS*.

6) For every allocated CLI descriptor area *AD* in *L2*:

   a) Let *DH* be the DescriptorHandle that identifies *AD*.

b) FreeHandle is implicitly invoked with HandleType indicating DESCRIPTOR HANDLE and with *DH* as the value of Handle.

NOTE 19 – Any diagnostic information generated by the invocation is associated with *C* and not with *AD*.

7) Let *CC* be the current SQL-connection.

8) The SQL-session associated with *EC* is terminated. *EC* is terminated, regardless of any exception conditions that might occur during the disconnection process, and is no longer associated with *C*.

9) If any error is detected during the disconnection process, then a completion condition is raised: *warning — disconnect error*.

10) If *EC* and *CC* were the same SQL-connection, then there is no current SQL-connection. Otherwise, *CC* remains the current SQL-connection.

# 6.15 EndTran

## Function

Terminate an SQL-transaction.

## Definition

```
EndTran (
       HandleType          IN    SMALLINT,
       Handle              IN    INTEGER,
       CompletionType      IN    SMALLINT )
       RETURNS SMALLINT
```

## General Rules

1) Let *HT* be the value of HandleType and let *H* be the value of Handle.

2) If *HT* is not one of the code values in Table 13, "Codes used for handle types", then an exception condition is raised: *CLI-specific condition — invalid handle*.

3) Case:

   a) If *HT* indicates STATEMENT HANDLE, then:

      i) If *H* does not identify an allocated SQL-statement, then an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Otherwise, an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

   b) If *HT* indicates DESCRIPTOR HANDLE, then:

      i) If *H* does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Otherwise, an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

   c) If *HT* indicates CONNECTION HANDLE, then:

      i) If *H* does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Otherwise:

         1) Let *C* be the allocated SQL-connection identified by *H*.

         2) The diagnostics area associated with *C* is emptied.

         3) If *C* has an associated established SQL-connection that is active, then let *L1* be a list containing *C*; otherwise, let *L1* be an empty list.

d) If *HT* indicates ENVIRONMENT HANDLE, then:

i) If *H* does not identify an allocated SQL-environment or if it identifies an allocated SQL-environment that is a skeleton SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

ii) Otherwise:

1) Let *E* be the allocated SQL-environment identified by *H*.

2) The diagnostics area associated with *E* is emptied.

3) Let *L* be a list of the allocated SQL-connections associated with *E*. Let *L1* be a list of the allocated SQL-connections in *L* that have an associated established SQL-connection that is active.

4) Let *CT* be the value of CompletionType.

5) If *CT* is not one of the code values in Table 14, "Codes used for transaction termination", then an exception condition is raised: *CLI-specific condition — invalid transaction operation code*.

6) If *L1* is empty, then no further rules of this Subclause are applied.

7) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent, then an exception condition is raised: *invalid transaction termination*.

8) Let *L2* be a list of the allocated SQL-statements associated with allocated SQL-connections in *L1*.

9) If any of the allocated SQL-statements in *L2* has an associated deferred parameter number, then an exception condition is raised: *CLI-specific condition — function sequence error*.

10) Let *L3* be a list of the open cursors associated with allocated SQL-statements in *L2*.

11) For every open cursor *OC* in *L3*:

a) Let *S* be the allocated SQL-statement with which *OC* is associated.

b) *OC* is placed in the closed state and its copy of the select source is destroyed.

c) Any fetched row associated with *S* is removed from association with *S*.

12) If *CT* indicates COMMIT, then:

a) For every temporary table associated with the current SQL-transaction that specifies the ON COMMIT DELETE option and that was updated by the current SQL-transaction, the invocation of EndTran with *CT* indicating COMMIT is effectively preceded by the execution of a <delete statement: searched> that specifies DELETE FROM *T*, where *T* is the <table name> of that temporary table.

b) The effects specified in the General Rules of Subclause 14.3, "<set constraints mode statement>", in ISO/IEC 9075:1992, occur as if the statement SET CONSTRAINTS ALL IMMEDIATE were executed.

c) Case:

i) If any constraint is not satisfied, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback—integrity constraint violation*.

ii) If any other error preventing commitment of the SQL-transaction has occurred, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback* with an implementation-defined subclass value.

iii) Otherwise, any changes to SQL-data or schemas that were made by the current SQL-transaction are made accessible to all concurrent and subsequent SQL-transactions.

d) The current SQL-transaction is terminated.

13) If *CT* indicates ROLLBACK, then:

a) Any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled.

b) The current SQL-transaction is terminated.

# 6.16 Error

## Function

Return diagnostic information.

## Definition

```
Error (
       EnvironmentHandle    IN    INTEGER,
       ConnectionHandle     IN    INTEGER,
       StatementHandle      IN    INTEGER,
       Sqlstate             OUT   CHARACTER(5),
       NativeError          OUT   INTEGER,
       MessageText          OUT   CHARACTER(L),
       BufferLength         IN    SMALLINT,
       TextLength           OUT   SMALLINT )
       RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Case:

   a) If StatementHandle identifies an allocated SQL-statement, then let *IH* be the value of StatementHandle and let *HT* be the code value for STATEMENT HANDLE from Table 13, "Codes used for handle types".

   b) If StatementHandle is zero and ConnectionHandle identifies an allocated SQL-connection, then let *IH* be the value of ConnectionHandle and let *HT* be the code value for CONNECTION HANDLE from Table 13, "Codes used for handle types".

   c) If ConnectionHandle is zero and EnvironmentHandle identifies an allocated SQL-environment, then let *IH* be the value of EnvironmentHandle and let *HT* be the code value for ENVIRONMENT HANDLE from Table 13, "Codes used for handle types".

   d) Otherwise, an exception condition is raised: *CLI-specific condition — invalid handle*.

2) Let *R* be the most recently executed CLI routine, other than Error, GetDiagField, or GetDiagRec, for which *IH* was passed as a value of an input handle.

   NOTE 20 – The GetDiagField, GetDiagRec and Error routines may cause exception or completion conditions to be raised, but do they not cause status records to be generated.

3) Let *N* be the number of status records generated by the execution of *R*. Let *AP* be the number of status records generated by the execution of *R* already processed by Error. If *N* is zero or *AP* equals *N* then a completion condition is raised: *no data*, Sqlstate is set to '00000', the values of NativeError, MessageText, and TextLength are set to implementation-dependent values, and no further rules of this Subclause are applied.

4) Let *SR* be the first status record generated by the execution of *R* not yet processed by Error. Let *RN* be the number of the status record *SR*. Information is retrieved by implicitly executing GetDiagRec as follows:

   ```
   GetDiagRec (HT, IH, RN, Sqlstate,
   ```

```
NativeError, MessageText, BufferLength, TextLength)
```

5) Add *SR* to the list of status records generated by the execution of *R* already processed by Error.

# 6.17 ExecDirect

## Function

Execute a statement directly.

## Definition

```
ExecDirect (
       StatementHandle   IN      INTEGER,
       StatementText     IN      CHARACTER(L),
       TextLength        IN      SMALLINT )
       RETURNS SMALLINT
```

where *L* is determined by the value of TextLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If an open cursor is associated with *S*, then an exception condition is raised: *invalid cursor state*.

3) Let *TL* be the value of TextLength.

4) Case:

   a) If *TL* is not negative, then let *L* be *TL*.

   b) If *TL* indicates NULL TERMINATED, then let *L* be the number of octets of StatementText that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *P* be the first *L* octets of StatementText.

6) If *P* is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then let *CN* be the cursor name referenced by *P*. Let *C* be the allocated SQL-connection with which *S* is associated. If *CN* is not the name of a cursor associated with another allocated SQL-statement associated with *C*, then an exception condition is raised: *invalid cursor name*.

7) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation*.

   a) *P* does not conform to the Format, Syntax Rules or Access Rules for a <preparable statement> or *P* is a <commit statement> or a <rollback statement>.

NOTE 21 – See Table 11, "SQL-statement character codes for use in a diagnostics area", for the list of <preparable statement>s.

b) *P* contains a <comment>.

c) *P* contains a <dynamic parameter specification> in an invalid position as determined by the rules specified in Subclause 17.6, "<prepare statement>", in ISO/IEC 9075:1992.

8) The data type of any <dynamic parameter specification> contained in *P* is determined by the rules specified in Subclause 17.6, "<prepare statement>", in ISO/IEC 9075:1992.

9) The following objects associated with *S* are destroyed:

a) Any prepared statement.

b) Any cursor.

c) Any select source.

If a cursor associated with *S* is destroyed, then so are any prepared statements that reference that cursor.

10) *P* is prepared and executed.

11) Case:

a) If *P* is a <dynamic select statement> or a <dynamic single row select statement>, then:

i) *P* becomes the select source associated with *S*.

ii) If there is no cursor name associated with *S*, then a unique implementation-dependent name that has the prefix 'SQLCUR' or the prefix 'SQL_CUR' becomes the cursor name associated with *S*.

iii) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied to 'DESCRIBE', *P*, and *S*, as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

iv) The General Rules of Subclause 5.3.2, "Implicit cursor", are applied to *P* and *S* as *SELECT SOURCE* and *ALLOCATED STATEMENT*, respectively.

b) Otherwise:

i) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied to 'DESCRIBE', *P*, and *S*, as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

ii) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied to 'EXECUTE', *P*, and *S*, as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

iii) Case:

1) If *P* is a <preparable dynamic delete statement: positioned>, then:

A) Let *CR* be the cursor referenced by *P*.

B) All the General Rules in Subclause 17.19, "<preparable dynamic delete statement: positioned>", in ISO/IEC 9075:1992, apply to *P*.

C) If the execution of *P* deleted the current row of *CR*, then the effect on the fetched row, if any, associated with the allocated SQL-statement under which that current row was established, is implementation-defined.

2) If *P* is a <preparable dynamic update statement: positioned>, then:

A) Let *CR* be the cursor referenced by *P*.

B) All the General Rules in Subclause 17.20, "<preparable dynamic update statement: positioned>", in ISO/IEC 9075:1992, apply to *P*.

C) If the execution of *P* updated the current row of *CR*, then the effect on the fetched row, if any, associated with the allocated SQL-statement under which that current row was established, is implementation-defined.

3) Otherwise, the results of the execution are the same as if the statement were contained in a <procedure> and executed; these are described in Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992.

12) Let *R* be the value of the ROW_COUNT field from the diagnostics area associated with *S*.

13) *R* becomes the row count associated with *S*.

14) If *P* executed successfully, then any executed statement associated with *S* is destroyed and *P* becomes the executed statement associated with *S*.

# 6.18 Execute

## Function

Execute a prepared statement.

## Definition

```
Execute (
       StatementHandle   IN    INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no prepared statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*. Otherwise, let *P* be the statement that was prepared.

3) If an open cursor is associated with *S*, then an exception condition is raised: *invalid cursor state*.

4) *P* is executed.

5) Case:

   a) If *P* is a <dynamic select statement> or a <dynamic single row select statement>, then the General Rules of Subclause 5.3.2, "Implicit cursor", are applied to *P* and *S* as SELECT SOURCE and ALLOCATED STATEMENT, respectively.

   b) Otherwise:

      i) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied to 'EXECUTE', *P*, and *S*, as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

      ii) Case:

         1) If *P* is a <preparable dynamic delete statement: positioned>, then:

            A) Let *CR* be the cursor referenced by *P*.

            B) All the General Rules in Subclause 17.19, "<preparable dynamic delete statement: positioned>", in ISO/IEC 9075:1992, apply to *P*.

            C) If the execution of *P* deleted the current row of *CR*, then the effect on the fetched row, if any, associated with the allocated SQL-statement under which that current row was established, is implementation-defined.

         2) If *P* is a <preparable dynamic update statement: positioned>, then:

            A) Let *CR* be the cursor referenced by *P*.

            B) All the General Rules in Subclause 17.20, "<preparable dynamic update statement: positioned>", in ISO/IEC 9075:1992, apply to *P*.

C) If the execution of *P* updated the current row of *CR*, then the effect on the fetched row, if any, associated with the allocated SQL-statement under which that current row was established, is implementation-defined.

3) Otherwise, the results of the execution are the same as if the statement were contained in a <procedure> and executed; these are described in Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992.

6) Let *R* be the value of the ROW_COUNT field from the diagnostics area associated with *S*.

7) *R* becomes the row count associated with *S*.

8) If *P* executed successfully, then any executed statement associated with *S* is destroyed and *P* becomes the executed statement associated with *S*.

# 6.19 Fetch

## Function

Fetch the next row of a cursor.

## Definition

```
Fetch (
       StatementHandle       IN        INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

3) If there is no open cursor associated with *S*, then an exception condition is raised: *invalid cursor state*. Otherwise, let *CR* be the open cursor associated with *S* and let *T* be the table associated with the open cursor.

4) Let *ARD* be the current application row descriptor for *S* and let *N* be the value of COUNT for *ARD*.

5) Within the first *N* item descriptor areas of ARD, refer to a <target specification> whose corresponding item descriptor area has a non-zero value of DATA_POINTER as a *bound target* and refer to the corresponding <select list> column as a *bound column*.

6) Let *IDA* be the item descriptor area of *ARD* corresponding to the *i*-th bound target and let *TT* be the value of TYPE for *IDA*.

7) If *TT* indicates DEFAULT, then:

   a) Let *IRD* be the implementation row descriptor associated with *S*.

   b) Let *CT*, *P*, and *SC* be the values of TYPE, PRECISION, and SCALE, respectively, for the item descriptor area of *IRD* corresponding to the *i*-th bound column.

   c) The data type, precision, and scale of the <target specification> described by *IDA* are effectively set to *CT*, *P*, and *SC*, respectively, for the purposes of this Fetch invocation only.

8) If *T* is empty or if the position of *CR* is on or after the last row of *T*, then:

   a) *CR* is positioned after the last row of *T*.

   b) A completion condition is raised: *no data*, an empty row becomes the fetched row associated with *S*, no database values are assigned to bound targets, and no further rules of this Subclause are applied.

9) Case:

   a) If the position of *CR* is before a row *NR*, then *CR* is positioned on row *NR*.

b) If the position of *CR* is on a row *OR* other than the last row, then *CR* is positioned on the row immediately after *OR*. Let *NR* be the row immediately after *OR*.

10) *NR* becomes the current row of *CR*.

11) Case:

a) If an exception condition is raised during derivation of any <derived column> associated with *NR*, then there is no fetched row associated with *S*, but *NR* remains the current row of *CR*.

b) Otherwise:

i) *NR* becomes the fetched row associated with *S*.

ii) Let *SS* be the select source associated with *S*.

iii) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied with 'FETCH', *SS*, and *S* as *TYPE*, *SOURCE* and *ALLOCATED STATEMENT*, respectively.

iv) If an exception condition is raised during the derivation of any target value, then the values of all the bound targets are implementation-dependent and *CR* remains positioned on the current row.

# 6.20 FetchScroll

## Function

Position a cursor on the specified row and retrieve values from that row.

## Definition

```
FetchScroll (
       StatementHandle          IN      INTEGER,
       FetchOrientation         IN      SMALLINT,
       FetchOffset              IN      INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

3) If there is no open cursor associated with *S*, then an exception condition is raised: *invalid cursor state*; otherwise, let *CR* be the open cursor associated with *S* and let *T* be the table associated with the open cursor.

4) If FetchOrientation is not one of the code values in Table 21, "Codes used for fetch orientation", then an exception condition is raised: *CLI-specific condition — invalid fetch orientation*.

5) Let *FO* be the value of FetchOrientation.

6) If the value of the CURSOR SCROLLABLE attribute of *S* is NONSCROLLABLE, and *FO* does not indicate NEXT, then an exception condition is raised: *CLI-specific condition — invalid fetch orientation*.

7) Let *ARD* be the current application row descriptor for *S* and let *N* be the value of COUNT for *ARD*.

8) Within the first *N* item descriptor areas of *ARD*, refer to a <target specification> whose corresponding item descriptor area has a non-zero value of DATA_POINTER as a *bound target* and refer to the corresponding <select list> column as a *bound column*.

9) Let *IDA* be the item descriptor area of *ARD* corresponding to the *i*-th bound target and let *TT* be the value of TYPE for *IDA*.

10) If *TT* indicates DEFAULT, then:

    a) Let *IRD* be the implementation row descriptor associated with *S*.

    b) Let *CT*, *P*, and *SC* be the values of TYPE, PRECISION, and SCALE, respectively, for the item descriptor area of *IRD* corresponding to the *i*-th bound column.

    c) The data type, precision, and scale of the <target specification> described by *IDA* are effectively set to *CT*, *P*, and *SC*, respectively, for the purposes of this fetch only.

11) Case:

a) If *FO* indicates ABSOLUTE or RELATIVE, then let *J* be the value of FetchOffset.

b) If *FO* indicates NEXT or FIRST, then let *J* be +1.

c) If *FO* indicates PRIOR or LAST, then let *J* be −1.

12) Let $T_t$ be a table of the same degree as *T*.

Case:

a) If *FO* indicates ABSOLUTE, FIRST, or LAST, then let $T_t$ contain all rows of *T*, preserving their order in *T*.

b) If *FO* indicates NEXT or indicates RELATIVE with a positive value of *J*, then:

i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or after the last row of *T*, then let $T_t$ be a table of no rows.

ii) If the position of *CR* is on a row *R* that is other than the last row of *T*, then let $T_t$ contain all rows of *T* ordered after row *R*, preserving their order in *T*.

iii) If the position of *CR* is before a row *R*, then let $T_t$ contain row *R* and all rows of *T* ordered after row *R*, preserving their order in *T*.

c) If *FO* indicates PRIOR or indicates RELATIVE with a negative value of *J*, then:

i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or before the first row of *T*, then let $T_t$ be a table of no rows.

ii) If the position of *CR* is on a row *R* that is other than the first row of *T*, then let $T_t$ contain all rows of *T* ordered before row *R*, preserving their order in *T*.

iii) If the position of *CR* is before the next row of a row *R* that is not the last row of *T*, then let $T_t$ contain row *R* and all rows of *T* ordered before row *R*, preserving their order in *T*.

iv) If the position of *CR* is after the last row of *T*, then let $T_t$ contain all rows of *T*, preserving their order in *T*.

d) If *FO* indicates RELATIVE with a zero value of *J*, then:

i) If the position of *CR* is on a row of *T*, then let $T_t$ be a table comprising that one row.

ii) Otherwise, let $T_t$ be an empty table.

13) Let *N* be the number of rows in $T_t$. If *J* is positive, then let *K* be *J*. If *J* is negative, then let *K* be $N + J + 1$. If *J* is zero, then let *K* be 1.

14) Case:

a) If *K* is greater than 0 and not greater than *N*, then *CR* is positioned on the *K*-th row of $T_t$ and the corresponding row of *T*. That row becomes the current row of *CR*. Let *NR* be that row.

b) Otherwise, no SQL-data values are assigned and a completion condition is raised: *no data*.

Case:

i) If *FO* indicates RELATIVE with *J* equal to zero, then the position of *CR* is unchanged.

ii) If *FO* indicates NEXT, indicates ABSOLUTE or RELATIVE with *K* greater than *N*, or indicates LAST, then *CR* is positioned after the last row.

iii) Otherwise, *FO* indicates PRIOR, FIRST, or ABSOLUTE or RELATIVE with *K* not greater than *N* and *CR* is positioned before the first row.

15) Case:

a) If a completion condition: *no data* has been raised, then no further rules of this Subclause are applied.

b) If an exception condition is raised during derivation of any <derived column> associated with *NR*, then there is no fetched row associated with *S*, but *NR* remains the current row of *CR*.

c) Otherwise:

i) *NR* becomes the fetched row associated with *S*.

ii) Let *SS* be the select source associated with *S*.

iii) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied with 'FETCH', *SS*, and *S* as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

iv) If an exception condition occurs during the derivation of any target value, then the values of all the bound targets are implementation-dependent and *CR* remains positioned on the current row.

# 6.21 FreeConnect

## Function

Deallocate an SQL-connection.

## Definition

```
FreeConnect (
      ConnectionHandle          IN   INTEGER )
      RETURNS SMALLINT
```

## General Rules

1) Let *CH* be the value of ConnectionHandle.

2) FreeHandle is implicitly invoked with HandleType indicating CONNECTION HANDLE and with *CH* as the value of Handle.

## 6.22 FreeEnv

### Function

Deallocate an SQL-environment.

### Definition

```
FreeEnv (
      EnvironmentHandle            IN    INTEGER )
      RETURNS SMALLINT
```

### General Rules

1) Let *EH* be the value of EnvironmentHandle.

2) FreeHandle is implicitly invoked with HandleType indicating ENVIRONMENT HANDLE and with *EH* as the value of Handle.

# 6.23 FreeHandle

## Function

Free a resource.

## Definition

```
FreeHandle (
       HandleType        IN    SMALLINT,
       Handle            IN    INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *HT* be the value of HandleType and let *H* be the value of Handle.

2) If *HT* is not one of the code values in Table 13, "Codes used for handle types", then an exception condition is raised: *CLI-specific condition — invalid handle*.

3) Case:

   a) If *HT* indicates ENVIRONMENT HANDLE, then:

      i) If *H* does not identify an allocated SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Let *E* be the allocated SQL-environment identified by *H*.

      iii) The diagnostics area associated with *E* is emptied.

      iv) If an allocated SQL-connection is associated with *E*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

      v) *E* is deallocated and all its resources are freed.

   b) If *HT* indicates CONNECTION HANDLE, then:

      i) If *H* does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Let *C* be the allocated SQL-connection identified by *H*.

      iii) The diagnostics area associated with *C* is emptied.

      iv) If an established SQL-connection is associated with *C*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

      v) *C* is deallocated and all its resources are freed.

   c) If *HT* indicates STATEMENT HANDLE, then:

      i) If *H* does not identify an allocated SQL-statement, then an exception condition is raised: *CLI-specific condition — invalid handle*.

      ii) Let *S* be the allocated SQL-statement identified by *H*.

iii) The diagnostics area associated with *S* is emptied.

iv) Let *C* be the allocated SQL-connection with which *S* is associated and let *EC* be the established SQL-connection associated with *C*.

v) 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.

vi) If there is a deferred parameter number associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

vii) If there is an open cursor associated with *S*, then:

1) The open cursor associated with *S* is placed in the closed state and its copy of the select source is destroyed.

2) Any fetched row associated with *S* is removed from association with *S*.

viii) The automatically allocated CLI descriptor areas associated with *S* are deallocated and all their resources are freed.

ix) *S* is deallocated and all its resources are freed.

d) If *HT* indicates DESCRIPTOR HANDLE, then:

i) If *H* does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*.

ii) Let *D* be the allocated CLI descriptor area identified by *H*.

iii) The diagnostics area associated with *D* is emptied.

iv) Let *C* be the allocated SQL-connection with which *D* is associated and let *EC* be the established SQL-connection associated with *C*.

v) 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.

vi) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *D* as the DESCRIPTOR AREA.

vii) Let *AT* be the value of the ALLOC_TYPE field of *D*.

viii) If *AT* indicates AUTOMATIC, then an exception condition is raised: *CLI-specific condition — invalid use of automatically-allocated descriptor handle*.

ix) Let *L1* be a list of allocated SQL-statements associated with *C* for which *D* is the current application row descriptor. For each allocated SQL-statement *S* in *L1*, the automatically-allocated application row descriptor associated with *S* becomes the current application row descriptor for *S*.

x) Let *L2* be a list of allocated SQL-statements associated with *C* for which *D* is the current application parameter descriptor. For each allocated SQL-statement *S* in *L2*, the automatically-allocated application parameter descriptor associated with *S* becomes the current application parameter descriptor for *S*.

xi) *D* is deallocated and all its resources are freed.

# 6.24 FreeStmt

## Function

Deallocate an SQL-statement.

## Definition

```
FreeStmt (
      StatementHandle           IN    INTEGER ,
      Option                    IN    INTEGER )
      RETURNS SMALLINT
```

## General Rules

1) Let *SH* be the value of StatementHandle and let *S* be the allocated SQL-statement identified by *SH*.

2) Let *OPT* be the value of Option.

3) If *OPT* is not one of the codes in Table 18, "Codes used for FreeStmt options", then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

4) Let *ARD* be the current application row descriptor for *S* and let *RC* be the value of COUNT for *ARD*.

5) Let *APD* be the current application parameter descriptor for *S* and let *PC* be the value of COUNT for *APD*.

6) Case:

   a) If *OPT* indicates CLOSE CURSOR and there is an open cursor associated with *S*, then:

      i) The open cursor associated with *S* is placed in the closed state and its copy of the select source is destroyed.

      ii) Any fetched row associated with *S* is removed from association with *S*.

   b) If *OPT* indicates FREE HANDLE, then FreeHandle is implicitly invoked with HandleType indicating STATEMENT HANDLE and with *SH* as the value of Handle.

   c) If *OPT* indicates UNBIND COLUMNS, then for each of the first *RC* item descriptor areas of *ARD*, the value of the DATA_POINTER field is set to zero.

   d) If *OPT* indicates UNBIND PARAMETERS, then for each of the first *PC* item descriptor areas of *APD*, the value of the DATA_POINTER field is set to zero.

# 6.25 GetConnectAttr

## Function

Get the value of an SQL-connection attribute.

## Definition

```
GetConnectAttr (
       ConnectionHandle      IN        INTEGER,
       Attribute             IN        INTEGER,
       Value                 OUT       ANY,
       BufferLength          IN        INTEGER,
       StringLength          OUT       INTEGER )
       RETURNS SMALLINT
```

NOTE 22 – The BufferLength and StringLength parameters are not used at present.

## General Rules

1) Case:

   a) If ConnectionHandle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *C* be the allocated SQL-connection identified by ConnectionHandle.

      ii) The diagnostics area associated with *C* is emptied.

2) Let *A* be the value of Attribute.

3) If *A* is not one of the code values in Table 16, "Codes used for connection attributes", then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

4) If *A* indicates POPULATE IPD, then:

   a) If there is no established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection does not exist*.

   b) Otherwise:

      i) If POPULATE IPD for *C* is *true* , then Value is set to 1.

      ii) If POPULATE IPD for *C* is *false* , then Value is set to 0.

# 6.26 GetCursorName

## Function

Get a cursor name.

## Definition

```
GetCursorName (
       StatementHandle   IN    INTEGER,
       CursorName        OUT   CHARACTER(L),
       BufferLength      IN    SMALLINT,
       NameLength        OUT   SMALLINT )
       RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Case:

   a) If there is no cursor name associated with *S*, then a unique implementation-dependent name that has the prefix 'SQLCUR' or the prefix 'SQL_CUR' becomes the cursor name associated with *S*; let *CN* be that associated cursor name.

   b) Otherwise, let *CN* be the cursor name associated with *S*.

   NOTE 23 – *CN* is an <identifier> and therefore may contain "<introducer><character set name>". See ISO/IEC 9075:1992, Subclause 5.4, "Names and identifiers".

3) Let *BL* be the value of BufferLength.

4) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with CursorName, *CN*, *BL*, and NameLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

# 6.27 GetData

## Function

Retrieve a column value.

## Definition

```
GetData (
       StatementHandle      IN      INTEGER,
       ColumnNumber         IN      SMALLINT,
       TargetType           IN      SMALLINT,
       TargetValue          OUT     ANY,
       BufferLength         IN      INTEGER,
       StrLen_or_Ind        OUT     INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Case:

   a) If there is no fetched row associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

   b) If the fetched row associated with *S* is empty, then a completion condition is raised: *no data*, TargetValue, StringLength, and StrLen_or_Ind are set to implementation-dependent values, and no further rules of this Subclause are applied.

   c) Otherwise, let *FR* be the fetched row associated with *S*.

3) Let *ARD* be the current application row descriptor for *S* and let *N* be the value of COUNT for *ARD*.

4) Let *D* be the degree of the table defined by the select source associated with *S*.

5) If *N* is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.

6) Let *CN* be the value of ColumnNumber.

7) If *CN* is less than 1 or greater than *D*, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

8) If DATA_POINTER is non-zero for at least one of the first *N* item descriptor areas of *ARD*, then let *BCN* be the column number associated with such an item descriptor area and let *HBCN* be the value of MAX(*BCN*). Otherwise, let *HBCN* be zero.

9) Let *IDA* be the item descriptor area of *ARD* specified by *CN*.

10) If *CN* is not greater than *HBCN*, then

Case:

a) If DATA_POINTER for *IDA* is not zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

b) If DATA_POINTER for *IDA* is zero, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

NOTE 24 – This implementation-defined feature determines whether columns before the highest bound column can be accessed by GetData.

11) If there is a fetched column number associated with *FR*, then let *FCN* be that column number; otherwise, let *FCN* be zero.

12) Case:

a) If *FCN* is greater than zero and *CN* is not greater than *FCN*, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

NOTE 25 – This implementation-defined feature determines whether GetData can only access columns in ascending column number order.

b) If *FCN* is less than zero, then:

i) Let *AFCN* be the absolute value of *FCN*.

ii) Case:

1) If *CN* is less than *AFCN*, then it is implementation-defined whether an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

NOTE 26 – This implementation-defined feature determines whether GetData can only access columns in ascending column number order.

2) If *CN* is greater than *AFCN*, then let *FCN* be *AFCN*.

13) Let *T* be the value of TargetType.

14) 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*.

15) If either of the following is true, then an exception condition is raised: *CLI-specific condition — invalid data type in application descriptor*.

a) *T* indicates neither DEFAULT nor ARD TYPE and is not one of the code values in Table 23, "Codes used for GetData data types".

b) *T* is one of the code values in Table 23, "Codes used for GetData data types", but the row that contains the corresponding SQL data type in the SQL data type column of the operative data type correspondence table contains 'None' in the host data type column.

16) If *T* does not indicate ARD TYPE, then the data type of the <target specification> described by *IDA* is set to *T*.

17) Let *IRD* be the implementation row descriptor associated with *S*.

18) If the value of TYPE for *IDA* indicates DEFAULT, then:

a) Let *CT*, *P*, and *SC* be the values of TYPE, PRECISION, and SCALE, respectively, for the *CN*-th item descriptor area of *IRD*.

b) The data type, precision, and scale of the <target specification> described by *IDA* are set to *CT*, *P*, and *S*, respectively, for the purposes of this GetData invocation only.

19) If *IDA* 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*.

20) Let *TT* be the value of TYPE for *IDA*.

21) Case:

a) If *TT* indicates CHARACTER, then:

i) Let *UT* be the code value corresponding to CHARACTER VARYING as specified in Table 6, "Codes used for implementation data types in CLI".

ii) Let *CL* be the implementation-defined maximum length for a CHARACTER VARYING data type.

b) Otherwise, let *UT* be *TT* and let *CL* be zero.

22) Case:

a) If *FCN* is less than zero, then

Case:

i) If *TT* does not indicate CHARACTER, then *AFCN* becomes the fetched column number associated with the fetched row associated with *S* and an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

ii) Otherwise, let *FL*, *DV*, and *DL* be the fetched length, data value and data length, respectively, associated with *FCN* and let *TV* be the result of the <string value function>:

```
SUBSTRING (DV FROM (FL+1))
```

b) Otherwise:

i) Let *FL* be zero.

ii) Let *SDT* be the effective data type of the *CN*-th <select list> 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 *CN*-th item descriptor area of *IRD*. Let *SV* be the value of the <select list> column, with data type *SDT*.

iii) Let *TDT* be the effective data type of the *CN*-th <target specification> as represented by the type *UT*, the length value *CL*, and the values of PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME for *IDA*.

iv) 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*.

v) 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.

vi) The <cast specification>

```
CAST (SV AS TDT)
```

is effectively performed, and is the value *TV* of the *CN*-th <target specification>.

23) *CN* becomes the fetched column number associated with the fetched row associated with *S*.

24) If *TV* is the null value, then

Case:

a) If StrLen_or_Ind is a null pointer, then an exception condition is raised: *data exception — null value, no indicator parameter*.

b) Otherwise, StrLen_or_Ind is set to −1.

25) Let *OL* be the value of BufferLength.

26) If null termination is *true* for the current SQL-environment, then let *NB* be the length in octets of a null terminator in the character set of the *i*-th bound target; otherwise let *NB* be 0.

27) If *TV* is not the null value, then:

a) StrLen_or_Ind is set to 0.

b) Case:

i) If *TT* does not indicate CHARACTER, then TargetValue is set to *TV*.

ii) Otherwise:

1) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with TargetValue, *TV*, *OL*, and StrLen_or_Ind as *TARGET*, *VALUE*, *OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

2) If *FCN* is not less than zero, then let *DV* be *TV* and let *DL* be the length of *TV* in octets.

3) Let *FL* be $(FL + OL - NB)$.

4) If *FL* is less than *DL*, then $-CN$ becomes the fetched column number associated with the fetched row associated with *S* and *FL*, *DV* and *DL* become the fetched length, data value, and data length, respectively, associated with the fetched column number.

# 6.28 GetDescField

## Function

Get a field from a CLI descriptor area.

## Definition

```
GetDescField (
      DescriptorHandle      IN      INTEGER,
      RecordNumber          IN      SMALLINT,
      FieldIdentifier       IN      SMALLINT,
      Value                 OUT     ANY,
      BufferLength          IN      INTEGER,
      StringLength          OUT     INTEGER )
      RETURNS SMALLINT
```

## General Rules

1) Let *D* be the allocated CLI descriptor area identified by DescriptorHandle and let *N* be the value of COUNT for *D*.

2) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *D* as the DESCRIPTOR AREA.

3) Let *FI* be the value of FieldIdentifier.

4) If *FI* is not one of the code values in Table 20, "Codes used for descriptor fields", then an exception condition is raised: *CLI-specific condition — invalid descriptor field identifier*.

5) Let *RN* be the value of RecordNumber.

6) If *FI* indicates neither COUNT nor ALLOC_TYPE, then:

   a) If *RN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

   b) If *RN* is greater than *N*, then a completion condition is raised: *no data*.

7) If *D* is an implementation row descriptor associated with an allocated SQL-statement *S* and there is no prepared or executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — associated statement is not prepared*.

8) Let *IDA* be the item descriptor area of *D* specified by *RN*.

9) Information is retrieved from *D*:

   Case:

   a) If *FI* indicates COUNT, then the value retrieved is *N*.

   b) If *FI* indicates ALLOC_TYPE, then the value retrieved is the allocation type for *D*.

   c) Otherwise, the value retrieved is the value of the field of *IDA* identified by *FI*.

10) Let *V* be the value retrieved.

11) Let *BL* be the value of BufferLength.

12) Case:

a) If the data type of *V* is character string, then the General Rules of Subclause 5.3.4, "Character string retrieval", are applied with Value, *V*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

b) Otherwise, Value is set to *V*.

# 6.29 GetDescRec

## Function

Get commonly-used fields from a CLI descriptor area.

## Definition

```
GetDescRec (
        DescriptorHandle        IN          INTEGER,
        RecordNumber            IN          SMALLINT,
        Name                    OUT         CHARACTER(L),
        BufferLength            IN          SMALLINT,
        NameLength              OUT         SMALLINT,
        Type                    OUT         SMALLINT,
        SubType                 OUT         SMALLINT,
        Length                  OUT         INTEGER,
        Precision               OUT         SMALLINT,
        Scale                   OUT         SMALLINT,
        Nullable                OUT         SMALLINT )
        RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *D* be the allocated CLI descriptor area identified by DescriptorHandle and let *N* be the value of COUNT for *D*.

2) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *D* as the DESCRIPTOR AREA.

3) Let *RN* be the value of RecordNumber.

4) Case:

   a) If *RN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

   b) Otherwise, if *RN* is greater than *N*, then a completion condition is raised: *no data*.

5) If *D* is an implementation row descriptor associated with an allocated SQL-statement *S* and there is no prepared or executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — associated statement is not prepared*.

6) Let *ITEM* be the <dynamic parameter specification> or <select list> column described by the item descriptor area of *D* specified by *RN*.

7) Let *BL* be the value of BufferLength.

8) Information is retrieved from *D*:

   a) If Type is not a null pointer, then Type is set to the data type of *ITEM*.

   b) If SubType is not a null pointer, then SubType is set to the datetime data type of *ITEM*.

c) If Length is not a null pointer, then Length is set to the length (in octets or positions, as appropriate) of *ITEM*.

d) If Precision is not a null pointer, then Precision is set to the precision of *ITEM*.

e) If Scale is not a null pointer, then Scale is set to the scale of *ITEM*.

f) If Nullable is not a null pointer, then Nullable is set to the nullable value of *ITEM*.

g) If Name is not a null pointer, then

Case:

i) If null termination is *false* for the current SQL-environment and *BL* is zero, then no further rules of this Subclause are applied.

ii) Otherwise:

1) The value retrieved is the name associated with *ITEM*.

2) Let *V* be the value retrieved.

3) The General Rules of Subclause 5.3.4, "Character string retrieval", are applied with Name, *V*, *BL*, and NameLength as *TARGET*, *VALUE*, *TARGET OCTET LENGTH*, and *RETURNED OCTET LENGTH*, respectively.

# 6.30 GetDiagField

## Function

Get information from a CLI diagnostics area.

## Definition

```
GetDiagField (
      HandleType            IN       SMALLINT,
      Handle                IN       INTEGER,
      RecordNumber          IN       SMALLINT,
      DiagIdentifier        IN       SMALLINT,
      DiagInfo              OUT      ANY,
      BufferLength          IN       SMALLINT,
      StringLength          OUT      SMALLINT )
      RETURNS SMALLINT
```

## General Rules

1) Let *HT* be the value of HandleType.

2) If *HT* is not one of the code values in Table 13, "Codes used for handle types", then an exception condition is raised: *CLI-specific condition — invalid handle*.

3) Case:

   a) If *HT* indicates ENVIRONMENT HANDLE and Handle does not identify an allocated SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) If *HT* indicates CONNECTION HANDLE and Handle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   c) If *HT* indicates STATEMENT HANDLE and Handle does not identify an allocated SQL-statement, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   d) If *HT* indicates DESCRIPTOR HANDLE and Handle does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*.

4) Let *DI* be the value of DiagIdentifier.

5) If *DI* is not one of the code values in Table 12, "Codes used for diagnostic fields", then an exception condition is raised: *CLI-specific condition — invalid attribute value*.

6) Case:

   a) If *DI* indicates MORE, ROW_COUNT, RETURNCODE, NUMBER, DYNAMIC_FUNCTION, or DYNAMIC_FUNCTION_CODE, then let *TYPE* be 'HEADER'.

   b) Otherwise, let *TYPE* be 'STATUS'.

7) Let *RN* be the value of RecordNumber.

8) Let *R* be the most recently executed CLI routine, other than GetDiagRec, GetDiagField, or Error, for which Handle was passed as the value of an input handle and let *N* be the number of status records generated by the execution of *R*.

NOTE 27 – The GetDiagRec, GetDiagField, and Error routines may cause exception or completion conditions to be raised, but they do not cause diagnostic information to be generated.

9) If *TYPE* is 'STATUS', then:

a) If *RN* is less than 1, then an exception condition is raised: *invalid condition number*.

b) If *RN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

10) If *DI* indicates ROW_COUNT and *R* is neither Execute nor ExecDirect, then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

11) If *TYPE* is 'HEADER', then header information from the diagnostics area associated with the resource identified by Handle is retrieved.

a) If *DI* indicates NUMBER, then the value retrieved is *N*.

b) If *DI* indicates DYNAMIC_FUNCTION, then

Case:

i) If no SQL-statement was being prepared or executed by *R*, then the value retrieved is a zero-length string.

ii) Otherwise, the value retrieved is the character identifier of the SQL-statement being prepared or executed by *R*. Table 11, "SQL-statement character codes for use in a diagnostics area", specifies the character identifier of the SQL-statement.

c) If *DI* indicates DYNAMIC_FUNCTION_CODE, then

Case:

i) If no SQL-statement was being prepared or executed by *R*, then the value retrieved is 0.

ii) Otherwise, the value retrieved is the integer identifier of the SQL-statement being prepared or executed by *R*. Table 10, "SQL-statement integer codes for use in a diagnostics area", specifies the integer identifier of the SQL-statement.

d) If *DI* indicates RETURNCODE, then the value retrieved is the code indicating the basic result of the execution of *R*. Subclause 4.2, "Return codes", specifies the code values and their meanings.

NOTE 28 – The value retrieved will never indicate Invalid handle or Data Needed , since no diagnostic information is generated if this is the basic result of the execution of *R*.

e) If *DI* indicates ROW_COUNT, the value retrieved is the number of rows affected as the result of executing a <delete statement: searched>, <insert statement> or <update statement: searched> as a direct result of the execution of the SQL-statement executed by *R*. Let *S* be the <delete statement: searched>, <insert statement> or <update statement: searched>. Let *T* be the table identified by the <table name> directly contained in *S*.

Case:

i) If *S* is an <insert statement>, then the value retrieved is the number of rows inserted into *T*.

ii) If *S* is not an <insert statement> and does not contain a <search condition>, then the value retrieved is the cardinality of *T* before the execution of *S*.

iii) Otherwise, let *SC* be the <search condition> directly contained in *S*. The value retrieved is effectively derived by executing the statement:

```
SELECT COUNT(*) FROM T WHERE SC
```

before the execution of *S*.

The value retrieved following the execution by *R* of an SQL-statement that does not directly result in the execution of a <delete statement: searched>, <insert statement> or <update statement: searched> is implementation-dependent.

f) If *DI* indicates MORE, then the value retrieved is:

i) If more conditions were raised during execution of *R* than have been stored in the diagnostics area, then 'Y'.

ii) If all the conditions that were raised during execution of *R* have been stored in the diagnostics area, then 'N'.

12) If *TYPE* is 'STATUS', then information from the *RN*-th status record in the diagnostics area associated with the resource identified by Handle is retrieved.

a) If *DI* indicates CONDITION_NUMBER, then the value retrieved is *RN*.

b) If *DI* indicates SQLSTATE, then the value retrieved is the SQLSTATE value corresponding to the status condition.

c) If *DI* indicates NATIVE_CODE, then the value retrieved is the implementation-defined native error code corresponding to the status condition.

d) If *DI* indicates MESSAGE_TEXT, then the value retrieved is an implementation-defined character string.

NOTE 29 – An implementation may provide <space>s or a zero-length string or a character string that describes the status condition.

e) If *DI* indicates MESSAGE_LENGTH, then the value retrieved is the length in characters of the character string value of MESSAGE_TEXT corresponding to the status condition.

f) If *DI* indicates MESSAGE_OCTET_LENGTH, then the value retrieved is the length in octets of the character string value of MESSAGE_TEXT corresponding to the status condition.

g) If *DI* indicates CLASS_ORIGIN, then the value retrieved is the identification of the naming authority that defined the class value of the SQLSTATE value corresponding to the status condition. That value shall be 'ISO 9075' if the class value is fully defined in Subclause 22.1, "SQLSTATE", in ISO/IEC 9075:1992, or Subclause 5.3.7, "CLI-specific status codes", and shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined class value.

h) If *DI* indicates SUBCLASS_ORIGIN, then the value retrieved is the identification of the naming authority that defined the subclass value of the SQLSTATE value corresponding to the status condition. That value shall be 'ISO 9075' if the subclass value is fully defined in Subclause 22.1, "SQLSTATE", in ISO/IEC 9075:1992, or Subclause 5.3.7, "CLI-specific status codes", and shall be an implementation-defined character string other than 'ISO 9075' for any implementation-defined subclass value.

i) If *DI* indicates CURSOR_NAME, CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME, CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, or COLUMN_NAME, then the values retrieved are:

i) If the value of SQLSTATE corresponds to *warning — cursor operation conflict*, then the value of CURSOR_NAME is the name of the cursor that caused the completion condition to be raised.

ii) If the value of SQLSTATE corresponds to *integrity constraint violation*, *transaction rollback — integrity constraint violation*, or *triggered data change violation*, then:

1) The values of CONSTRAINT_CATALOG and CONSTRAINT_SCHEMA are the <catalog name> and the <unqualified schema name> of the <schema name> of the schema containing the constraint or assertion. The value of CONSTRAINT_NAME is the <qualified identifier> of the constraint or assertion.

2) Case:

A) If the violated integrity constraint is a table constraint, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> or <local table name>, respectively, of the table in which the table constraint is contained.

B) If the violated integrity constraint is an assertion and if only one table referenced by the assertion has been modified as a result of executing the SQL-statement, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> or <local table name>, respectively, of the modified table.

C) Otherwise, the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are <space>s.

3) If TABLE_NAME identifies a declared local temporary table, then CATALOG_NAME is <space>s and SCHEMA_NAME is 'MODULE'.

iii) If the value of SQLSTATE corresponds to *syntax error or access rule violation*, then:

1) The values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the table that caused the syntax error or the access rule violation and the <qualified identifier> or <local table name>, respectively. If TABLE_NAME refers to a declared local temporary table, then CATALOG_NAME is <space>s and SCHEMA_NAME contains "MODULE".

2) If the syntax error or the access rule violation was for an inaccessible column, then the value of COLUMN_NAME is the <column name> of that column. Otherwise, the value of COLUMN_NAME is <space>s.

iv) If the value of SQLSTATE corresponds to *invalid cursor state*, then the value of CURSOR_NAME is the name of the cursor that is in the invalid state.

v) If the value of SQLSTATE corresponds to *with check option violation*, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name> and the <unqualified schema name> of the <schema name> of the schema that contains the view that caused the violation of the WITH CHECK OPTION, and the <qualified identifier> of that view, respectively.

vi) If the value of SQLSTATE does not correspond to *syntax error or access rule violation*, then:

1) If the values of CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, and COLUMN_NAME identify a column for which no privileges are granted to the current <authorization identifier>, then the value of COLUMN_NAME is replaced by a zero-length string.

2) If the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME identify a table for which no privileges are granted to the current <authorization identifier>, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are replaced by a zero-length string.

3) If the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME identify a <table constraint> for some table *T* and if no privileges for *T* are granted to the current <authorization identifier>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are replaced by a zero-length string.

4) If the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME identify an assertion contained in some schema *S* and if the owner of *S* is not the <authorization identifier> of the current SQL-session, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are replaced by a zero-length string.

j) If *DI* indicates SERVER_NAME or CONNECTION_NAME, then the values retrieved are

Case:

i) If *R* is Connect, then the name of the SQL-server explicitly or implicitly referenced by *R* and the implementation-defined connection name associated with that SQL-server reference, respectively.

ii) If *R* is Disconnect, then the name of the SQL-server and the associated implementation-defined connection name, respectively, associated with the allocated SQL-connection referenced by *R*.

iii) If the status condition was caused by the application of the General Rules of Subclause 5.3.1, "Implicit set connection", then the name of the SQL-server and the implementation-defined connection name, respectively, associated with the dormant connection specified in the application of that Subclause.

iv) If the status condition was raised in an SQL-session, then the name of the SQL-server and the implementation-defined connection name, respectively, associated with the SQL-session in which the status condition was raised.

v) Otherwise, zero-length strings.

13) Let *V* be the value retrieved.

14) If the data type of *V* is not character string, then DiagInfo is set to *V* and no further rules of this Subclause are applied.

15) Let *BL* be the value of BufferLength.

16) If *BL* is not greater than zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

17) Let *L* be the length in octets of *V*.

18) If StringLength is not a null pointer, then StringLength is set to *L*.

19) Case:

a) If null termination is *false* for the current SQL-environment, then:

i) If *L* is not greater than *BL*, then the first *L* octets of DiagInfo are set to *V* and the values of the remaining octets of DiagInfo are implementation-dependent.

ii) Otherwise, DiagInfo is set to the first *BL* octets of *V*.

b) Otherwise, let *k* be the number of octets in a null terminator in the character set of DiagInfo and let the phrase "implementation-defined null character that terminates a C character string" imply *k* octets, all of whose bits are zero.

i) If *L* is not greater than (*BL*−*k*), then the first (*L*+*k*) octets of DiagInfo 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 DiagInfo are implementation-dependent.

ii) Otherwise, DiagInfo is set to the first (*BL*−*k*) octets of *V* concatenated with a single implementation-defined null character that terminates a C character string.

# 6.31 GetDiagRec

## Function

Get commonly-used information from a CLI diagnostics area.

## Definition

```
GetDiagRec (
      HandleType            IN      SMALLINT,
      Handle                IN      INTEGER,
      RecordNumber          IN      SMALLINT,
      Sqlstate              OUT     CHARACTER(5),
      NativeError           OUT     INTEGER,
      MessageText           OUT     CHARACTER(L),
      BufferLength          IN      SMALLINT,
      TextLength            OUT     SMALLINT )
      RETURNS SMALLINT
```

where *L* is the value of BufferLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *HT* be the value of HandleType.

2) If *HT* is not one of the code values in Table 13, "Codes used for handle types", then an exception condition is raised: *CLI-specific condition — invalid handle*.

3) Case:

   a) If *HT* indicates ENVIRONMENT HANDLE and Handle does not identify an allocated SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) If *HT* indicates CONNECTION HANDLE and Handle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   c) If *HT* indicates STATEMENT HANDLE and Handle does not identify an allocated SQL-statement, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   d) If *HT* indicates DESCRIPTOR HANDLE and Handle does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid handle*.

4) Let *RN* be the value of RecordNumber.

5) Let *R* be the most recently executed CLI routine, other than GetDiagRec, GetDiagField, or Error, for which Handle was passed as the value of an input handle and let *N* be the number of status records generated by the execution of *R*.

   NOTE 30 – The GetDiagRec, GetDiagField, and Error routines may cause exception or completion conditions to be raised, but they do not cause diagnostic information to be generated.

6) If *RN* is less than 1, then an exception condition is raised: *invalid condition number*.

7) If *RN* is greater than *N*, then a completion condition is raised: *no data*, and no further rules of this Subclause are applied.

8) Let *BL* be the value of BufferLength.

9) Information from the *RN*-th status record in the diagnostics area associated with the resource identified by Handle is retrieved.

   a) If Sqlstate is not a null pointer, then Sqlstate is set to the SQLSTATE value corresponding to the status condition.

   b) If NativeError is not a null pointer, then NativeError is set to the implementation-defined native error code corresponding to the status condition.

   c) If MessageText is not a null pointer, then

      Case:

      i) If null termination is *false* for the current SQL-environment and *BL* is zero, then no further rules of this Subclause are applied.

      ii) Otherwise, an implementation-defined character string is retrieved. Let *MT* be the implementation-defined character string that is retrieved and let *L* be the length in octets of *MT*. If *BL* is not greater than zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*. If TextLength is not a null pointer, then TextLength is set to *L*.

         Case:

         1) If null termination is *false* for the current SQL-environment, then:

            A) If *L* is not greater than *BL*, then the first *L* octets of MessageText are set to *V* and the values of the remaining octets of MessageText are implementation-dependent.

            B) Otherwise, MessageText is set to the first *BL* octets of *V*.

         2) Otherwise, let *k* the number of octets in a null terminator in the character set of MessageText and let the phrase "implementation-defined null character that terminates a C character string" imply *k* octets, all of whose bits are zero.

            A) If *L* is not greater than (*BL*−*k*), then the first (*L*+*k*) octets of MessageText 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 MessageText are implementation-dependent.

            B) Otherwise, MessageText is set to the first (*BL*−*k*) octets of *V* concatenated with a single implementation-defined null character that terminates a C character string.

      NOTE 31 – An implementation may provide <space>s or a zero-length string or a character string that describes the status condition.

# 6.32 GetEnvAttr

## Function

Get the value of an SQL-environment attribute.

## Definition

```
GetEnvAttr (
       EnvironmentHandle     IN        INTEGER,
       Attribute             IN        INTEGER,
       Value                 OUT       ANY,
       BufferLength          IN        INTEGER,
       StringLength          OUT       INTEGER )
       RETURNS SMALLINT
```

NOTE 32 – The BufferLength and StringLength parameters are not used at present.

## General Rules

1) Case:

   a) If EnvironmentHandle does not identify an allocated SQL-environment or if it identifies an allocated skeleton SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *E* be the allocated SQL-environment identified by EnvironmentHandle.

      ii) The diagnostics area associated with *E* is emptied.

2) Let *A* be the value of Attribute.

3) If *A* is not one of the code values in Table 15, "Codes used for environment attributes", then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

4) If *A* indicates NULL TERMINATION, then

   Case:

   a) If null termination for *E* is *true* , then Value is set to 1.

   b) If null termination for *E* is *false* , then Value is set to 0.

# 6.33 GetFunctions

## Function

Determine whether a CLI routine is supported.

## Definition

```
GetFunctions (
       ConnectionHandle          IN       INTEGER,
       FunctionId                IN       SMALLINT,
       Supported                 OUT      SMALLINT )
       RETURNS SMALLINT
```

## General Rules

1) Case:

   a) If ConnectionHandle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *C* be the allocated SQL-connection identified by ConnectionHandle.

      ii) The diagnostics area associated with *C* is emptied.

2) Case:

   a) If there is no established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection does not exist*.

   b) Otherwise, let *EC* be the established SQL-connection associated with *C*.

3) If *EC* is not the current SQL-connection, then the General Rules of Subclause 5.3.1, "Implicit set connection", are applied to *EC* as the dormant SQL-connection.

4) Let *FI* be the value of FunctionId.

5) If *FI* identifies a CLI routine that is supported by the implementation, then Supported is set to 1; otherwise, Supported is set to 0. Table 24, "Codes used to identify SQL/CLI routines", specifies the codes used to identify the CLI routines defined in ISO/IEC 9075.

# 6.34 GetInfo

## Function

Get information about the implementation.

## Definition

```
GetInfo (
       ConnectionHandle          IN       INTEGER,
       InfoType                  IN       SMALLINT,
       InfoValue                 OUT      ANY,
       BufferLength              IN       INTEGER,
       StringLength              OUT      INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Case:

   a) If ConnectionHandle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *C* be the allocated SQL-connection identified by ConnectionHandle.

      ii) The diagnostics area associated with *C* is emptied.

2) Case:

   a) If there is no established SQL-connection associated with *C*, then an exception condition is raised: *connection exception — connection does not exist*.

   b) Otherwise, let *EC* be the established SQL-connection associated with *C*.

3) If *EC* is not the current SQL-connection, then the General Rules of Subclause 5.3.1, "Implicit set connection", are applied to *EC* as the dormant SQL-connection.

4) Let *IT* be the value of InfoType.

5) If *IT* is not one of the codes in Table 25, "Codes and data types for implementation information", then an exception condition is raised: *CLI-specific condition — invalid information type*.

6) Let *SS* be the SQL-server associated with *EC*.

7) Refer to a component of the SQL-client that is responsible for communicating with one or more SQL-servers as a driver.

8) Let *D* be the driver responsible for communicating with *SS*.

9) Case:

   a) If *IT* indicates MAXIMUM DRIVER CONNECTIONS, then let *V* be the maximum number of SQL-connections that may be concurrently established to *SS* through *D*. If the maximum number is not a known fixed number, then let *V* be 0.

b) If *IT* indicates MAXIMUM CONCURRENT ACTIVITIES then:

i) Let *AS* refer to an allocated SQL-statement. *AS* is said to be *currently active* if one of the following is true:

1) There is an open cursor associated with *AS*.

2) There is a deferred parameter number associated with *AS*.

ii) Let *V* be the maximum number of SQL-statements that may be currently active for an established SQL-connection to *SS* through *D*. If the maximum number is not a known fixed number, then let *V* be 0.

c) If *IT* indicates DBMS NAME, then let *V* be the implementation-defined character string that identifies the software supporting the functionality of *SS*.

d) If *IT* indicates DBMS VERSION, then let *V* be the character string that identifies the version number of the software supporting the functionality of *SS*. The value of the character string is implementation-defined, but the format of the value is:

```
<first part><period><second part><period><third part> [ <fourth part> ]
```

where:

```
<first part> ::= <digit><digit>
<second part> ::= <digit><digit>
<third part> ::= <digit><digit><digit><digit>
<fourth part> ::= <character representation>...
```

e) If *IT* indicates MAXIMUM COLUMN NAME LENGTH, MAXIMUM CURSOR NAME LENGTH, or MAXIMUM TABLE NAME LENGTH, then:

i) Let the length of an <identifier> be the number of <delimited identifier part>s if the identifier is a <delimited identifier> or the sum of the number of <identifier start>s, the number of <underscore>s, and the number of <identifier part>s if the <identifier> is a <regular identifier>.

ii) Case:

1) If *IT* indicates MAXIMUM COLUMN NAME LENGTH, then let *V* be the maximum length in characters supported by *SS* for an <identifier> that is a <column name>.

2) If *IT* indicates MAXIMUM CURSOR NAME LENGTH, then let *V* be the maximum length in characters supported by *SS* for an <identifier> that is a <cursor name>.

3) If *IT* indicates MAXIMUM TABLE NAME LENGTH, then let *V* be the maximum length in characters supported by *SS* for an <identifier> that is the <qualified identifier> of a <table name>.

4) If *IT* indicates MAXIMUM CATALOG NAME LENGTH, then let *V* be the maximum length in characters supported by *SS* for an <identifier> that is a <catalog name>.

5) If *IT* indicates MAXIMUM SCHEMA NAME LENGTH, then let *V* be the maximum length in characters supported by *SS* for an <identifier> that is the <unqualified schema name> of a <schema name>.

6) If *IT* indicates MAXIMUM IDENTIFIER LENGTH, then let *V* be the maximum length in characters supported by *SS* for an <identifier>. If *SS* supports different maximum lengths for different types of objects, then let *V* be the minimum of such maximum lengths.

f) If *IT* indicates ALTER TABLE, then:

i) Let *V* be 0.

ii) If *SS* supports the <add column definition> clause on the <alter table statement>, then add the numeric value for ADD COLUMN from Table 26, "Values for ALTER TABLE with GetInfo" to *V*.

iii) If *SS* supports the <drop column definition> clause on the <alter table statement>, then add the numeric value for DROP COLUMN from Table 26, "Values for ALTER TABLE with GetInfo" to *V*.

iv) If *SS* supports the <alter column definition> clause on the <alter table statement>, then add the numeric value for ALTER COLUMN from Table 26, "Values for ALTER TABLE with GetInfo" to *V*.

v) If *SS* supports the <add table constraint definition> clause on the <alter table statement>, then add the numeric value for ADD CONSTRAINT from Table 26, "Values for ALTER TABLE with GetInfo" to *V*.

vi) If *SS* supports the <drop table constraint definition> clause on the <alter table statement>, then add the numeric value for DROP CONSTRAINT from Table 26, "Values for ALTER TABLE with GetInfo" to *V*.

g) If *IT* indicates CATALOG NAME, then:

i) If *SS* supports <catalog name>s, then let *V* be 'Y'.

ii) Otherwise, then let *V* be 'N'.

h) If *IT* indicates COLLATING SEQUENCE, then let *V* be the character string that identifies the default collation for *SS*.

i) If *IT* indicates CURSOR COMMIT BEHAVIOR, then let *V* be one of the following values to indicate the default treatment that *SS* has for cursors and prepared statements when a transaction is committed:

i) If *SS* closes cursors and deletes prepared statements, then let *V* be the value for DELETE from Table 27, "Values for CURSOR COMMIT BEHAVIOR with GetInfo".

ii) If *SS* closes cursors and retains prepared statements, then let *V* be the value for CLOSE from Table 27, "Values for CURSOR COMMIT BEHAVIOR with GetInfo".

iii) If *SS* retains both cursors and prepared statements, then let *V* be the value for PRESERVE from Table 27, "Values for CURSOR COMMIT BEHAVIOR with GetInfo".

j) If *IT* indicates CURSOR SENSITIVITY, then let *V* be set as follows to indicate support for cursor sensitivity at *SS*:

i) If *SS* supports the behavior associated with the INSENSITIVE keyword in Clause 13, "Data manipulation", in ISO/IEC 9075:1992, then let *V* be the value of INSENSITIVE

for the CURSOR SENSITIVITY attribute from Table 22, "Miscellaneous codes used in CLI".

ii) If *SS* does not support the behavior associated with the INSENSITIVE keyword in Clause 13, "Data manipulation", in ISO/IEC 9075:1992, then let *V* be the value of UNSPECIFIED for the CURSOR SENSITIVITY attribute from Table 22, "Miscellaneous codes used in CLI".

k) If *IT* indicates DATA SOURCE NAME, then let *V* be the name that was used as the ServerName argument in the Connect routine that established *EC*.

l) If *IT* indicates DATA SOURCE READ ONLY, then:

i) If the data from *SS* can be read but not modified, then let *V* be 'Y'.

ii) Otherwise, let *V* be 'N'.

m) If *IT* indicates DEFAULT TRANSACTION ISOLATION, then let *V* be the appropriate value from Table 34, "Values for TRANSACTION ISOLATION OPTION with GetInfo", to indicate the default isolation level of *SS*.

n) If *IT* indicates DESCRIBE PARAMETER, then:

i) If *SS* is capable of describing dynamic parameters, then let *V* be 'Y'.

ii) Otherwise, let *V* be 'N'.

o) If *IT* indicates FETCH DIRECTION, then:

i) Let *V* be 0.

ii) If *SS* supports the behavior specified in Subclause 13.3, "<fetch statement>", in ISO/IEC 9075:1992, associated with a <fetch orientation> that specifies ABSOLUTE, then add the numeric value for FETCH ABSOLUTE from Table 28, "Values for FETCH DIRECTION with GetInfo", to *V*.

iii) If *SS* supports the behavior specified in Subclause 13.3, "<fetch statement>", in ISO/IEC 9075:1992, associated with a <fetch orientation> that specifies FIRST, then add the numeric value for FETCH FIRST from Table 28, "Values for FETCH DIRECTION with GetInfo", to *V*.

iv) If *SS* supports the behavior specified in Subclause 13.3, "<fetch statement>", in ISO/IEC 9075:1992, associated with a <fetch orientation> that specifies LAST, then add the numeric value for FETCH LAST from Table 28, "Values for FETCH DIRECTION with GetInfo", to *V*.

v) If *SS* supports the behavior specified in Subclause 13.3, "<fetch statement>", in ISO/IEC 9075:1992, associated with a <fetch orientation> that specifies NEXT, then add the numeric value for FETCH NEXT from Table 28, "Values for FETCH DIRECTION with GetInfo", to *V*.

vi) If *SS* supports the behavior specified in Subclause 13.3, "<fetch statement>", in ISO/IEC 9075:1992, associated with a <fetch orientation> that specifies PRIOR, then add the numeric value for FETCH PRIOR from Table 28, "Values for FETCH DIRECTION with GetInfo", to *V*.

vii) If *SS* supports the behavior specified in Subclause 13.3, "<fetch statement>", in ISO/IEC 9075:1992, associated with a <fetch orientation> that specifies RELATIVE, then add the numeric value for FETCH RELATIVE from Table 28, "Values for FETCH DIRECTION with GetInfo", to *V*.

p) If *IT* indicates GETDATA EXTENSIONS, then *V* is set as follows to indicate whether the implementation supports certain extensions to the GetData routine:

i) Let *V* be 0.

ii) If GetData can be called to obtain columns that precede the last bound column, then add the numeric value for ANY COLUMN from Table 29, "Values for GETDATA EXTENSIONS with GetInfo", to *V*.

iii) If GetData can be called for columns in any order, then add the numeric value for ANY ORDER from Table 29, "Values for GETDATA EXTENSIONS with GetInfo", to *V*.

NOTE 33 – This also means that a column can be accessed by GetData even though previous calls retrieved all the data for that column.

q) If *IT* indicates IDENTIFIER CASE, then let *V* be one of the following values to indicate what *SS* does when placing a non-delimited identifier *ID* into the catalog.

i) If *SS* stores the value of *ID* in upper case, then set *V* to the value for UPPER from Table 30, "Values for IDENTIFIER CASE with GetInfo".

ii) If *SS* stores the value of *ID* in lower case, then set *V* to the value for LOWER from Table 30, "Values for IDENTIFIER CASE with GetInfo".

iii) If *SS* stores the value of *ID* in mixed case, then set *V* to the value for SENSITIVE from Table 30, "Values for IDENTIFIER CASE with GetInfo".

iv) If *SS* stores the value of *ID* in mixed case but treats the value as case-insensitive when used, then set *V* to the value for MIXED from Table 30, "Values for IDENTIFIER CASE with GetInfo".

r) If *IT* indicates MAXIMUM COLUMNS IN GROUP BY, then let *V* be the maximum number of columns that *SS* supports in a GROUP BY clause. If there is no maximum, then let *V* be 0.

s) If *IT* indicates MAXIMUM COLUMNS IN ORDER BY, then let *V* be the maximum number of columns that *SS* supports in a ORDER BY clause. If there is no maximum, then let *V* be 0.

t) If *IT* indicates MAXIMUM COLUMNS IN SELECT, then let *V* be the maximum number of columns that *SS* supports in a <select list>. If there is no maximum, then let *V* be 0.

u) If *IT* indicates MAXIMUM COLUMNS IN TABLE, then let *V* be the maximum number of columns that *SS* supports in a base table. If there is no maximum, then let *V* be 0.

v) If *IT* indicates MAXIMUM STATEMENT LENGTH, then let *V* be the maximum number of characters that *SS* supports for an SQL statement. If there is no maximum, then let *V* be 0.

w) If *IT* indicates MAXIMUM TABLES IN SELECT, then let *V* be the maximum number of table names supported by *SS* in the FROM clause of a <query specification>. If there is no maximum, then let *V* be 0.

x) If *IT* indicates MAXIMUM USER NAME LENGTH, then let *V* be the maximum size in characters that *SS* supports for a user identifier. If this value is unknown, then let *V* be 0.

y) If *IT* indicates NULL COLLATION, then:

i) If *SS* sorts null values as greater than all non-null values, then let *V* be the value of HIGH from Table 35, "Values for NULL COLLATION with GetInfo".

ii) If *SS* sorts null values as less than all non-null values, then let *V* be the value of LOW from Table 35, "Values for NULL COLLATION with GetInfo".

z) If *IT* indicates OUTER JOIN CAPABILITIES, then:

i) Let *V* be 0.

ii) If *SS* supports the behavior for an <outer join type> specified as LEFT as specified in Subclause 7.5, "<joined table>", in ISO/IEC 9075:1992, then add the numeric value for LEFT from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

iii) If *SS* supports the behavior for an <outer join type> specified as RIGHT as specified in Subclause 7.5, "<joined table>", in ISO/IEC 9075:1992, then add the numeric value for RIGHT from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

iv) If *SS* supports the behavior for an <outer join type> specified as FULL as specified in Subclause 7.5, "<joined table>", in ISO/IEC 9075:1992, then add the numeric value for FULL from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

v) If *SS* supports nested outer joins, then add the numeric value for NESTED from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

vi) If *SS* supports join operations where the order of tables in the ON clause need not be the same as the order of the tables within the associated JOIN clause, then add the numeric value for NOT ORDERED from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

vii) If *SS* permits an inner table to be the result of a INNER JOIN, then add the numeric value for INNER from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

viii) If *SS* supports any predicate within an ON clause of an OUTER JOIN, then add the numeric value for ALL COMPARISON OPS from Table 31, "Values for OUTER JOIN CAPABILITIES with GetInfo", to *V*.

aa) If *IT* indicates ORDER BY COLUMNS IN SELECT, then:

i) If *SS* requires that columns in the ORDER BY clause also appear in the associated <select list>, then let *V* be 'Y'.

ii) Otherwise, let *V* be 'N'.

bb) If *IT* indicates SCROLL CONCURRENCY, then:

i) Let *V* be 0.

ii) If *SS* supports read-only scrollable cursors, then add the numeric value for READ ONLY from Table 32, "Values for SCROLL CONCURRENCY with GetInfo", to *V*.

iii) If *SS* supports scrollable cursors and allows this with the lowest level of locking that ensures that the row can be updated, then add the numeric value for LOCK from Table 32, "Values for SCROLL CONCURRENCY with GetInfo", to *V*.

iv) If *SS* supports scrollable cursors and allows this with optimistic concurrency using row identifiers or timestamps, add the numeric value for OPT ROWVER from Table 32, "Values for SCROLL CONCURRENCY with GetInfo", to *V*.

v) If *SS* supports scrollable cursors and allows this with optimistic concurrency by comparing values, add the numeric value for OPT VALUES from Table 32, "Values for SCROLL CONCURRENCY with GetInfo", to *V*.

cc) If *IT* indicates SERVER NAME, then let *V* be the implementation-defined character string that is the actual name of *SS*.

dd) If *IT* indicates SPECIAL CHARACTERS, then let *V* be a character string that contains all the characters other than upper case letters, lower case letters, digits, and underscore that *SS* allows in non-delimited identifiers. If *SS* allows no such characters, then let *V* be the empty string.

ee) If *IT* indicates TRANSACTION CAPABLE, then:

i) Let *DDL* represent the statements <alter domain statement>, <alter table statement>, <assertion definition>, <character set definition>, <collation definition>, <domain definition>, <drop assertion statement>, <drop character set statement>, <drop collation statement>, <drop domain statement>, <drop schema statement>, <drop table statement>, <grant statement>, <revoke statement>, <schema definition>, <table definition>, and <translation definition>.

ii) Let *DML* represent the statements listed in Table 10, "SQL-statement integer codes for use in a diagnostics area", that are not *DDL*.

iii) Let *V* be one of the following values in order to indicate the transaction-related restrictions that *SS* imposes on *DDL*.

1) If transactions can contain both *DML* and *DDL* without restriction, then let *V* be the value for ALL from Table 33, "Values for TRANSACTION CAPABLE with GetInfo".

2) If transactions can contain only *DML* and execution of a *DDL* statement during a transaction causes completion of that transaction before the processing of the *DDL* statement, then let *V* be the value for DDL COMMIT from Table 33, "Values for TRANSACTION CAPABLE with GetInfo".

3) If transactions can contain only *DML* and any *DDL* statement is ignored, then let *V* be the value for DDL IGNORE from Table 33, "Values for TRANSACTION CAPABLE with GetInfo".

4) If transactions can contain only *DML* and any *DDL* results in an error, then let *V* be the value for DML from Table 33, "Values for TRANSACTION CAPABLE with GetInfo".

5) If transactions are not supported at all and each statement is self-committing, then let *V* be the value for NONE from Table 33, "Values for TRANSACTION CAPABLE with GetInfo".

ff) If *IT* indicates TRANSACTION ISOLATION OPTION, then:

i) Let *V* be 0.

ii) If *SS* supports the READ UNCOMMITTED isolation level, then add the numeric value for READ UNCOMMITTED from Table 34, "Values for TRANSACTION ISOLATION OPTION with GetInfo", to V.

iii) If *SS* supports the READ COMMITTED isolation level, then add the numeric value for READ COMMITTED from Table 34, "Values for TRANSACTION ISOLATION OPTION with GetInfo", to V.

iv) If *SS* supports the REPEATABLE READ isolation level, then add the numeric value for REPEATABLE READ from Table 34, "Values for TRANSACTION ISOLATION OPTION with GetInfo", to V.

v) If *SS* supports the SERIALIZABLE isolation level, then add the numeric value for SERIALIZABLE from Table 34, "Values for TRANSACTION ISOLATION OPTION with GetInfo", to V.

gg) If *IT* indicates USER NAME, then let *V* be the value of CURRENT_USER.

10) Let *BL* be the value of BufferLength in octets.

11) Case:

a) If the data type of *V* is character string, then the General Rules of Subclause 5.3.4, "Character string retrieval", are applied with InfoValue, *V*, *BL*, and StringLength as *TARGET*, *VALUE*, *TARGET LENGTH*, and *RETURNED LENGTH*, respectively.

b) Otherwise, InfoValue is set to *V*.

# 6.35 GetStmtAttr

## Function

Get the value of an SQL-statement attribute.

## Definition

```
GetStmtAttr (
       StatementHandle       IN        INTEGER,
       Attribute             IN        INTEGER,
       Value                 OUT       ANY,
       BufferLength          IN        INTEGER,
       StringLength          OUT       INTEGER )
       RETURNS SMALLINT
```

NOTE 34 – The BufferLength and StringLength parameters are not used at present.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Let *A* be the value of Attribute.

3) If *A* is not one of the code values in Table 17, "Codes used for statement attributes", then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

4) Case:

   a) If *A* indicates APD_HANDLE, then Value is set to the handle of the current application parameter descriptor for *S*.

   b) If *A* indicates ARD_HANDLE, then Value is set to the handle of the current application row descriptor for *S*.

   c) If *A* indicates IPD_HANDLE, then Value is set to the handle of the implementation parameter descriptor associated with *S*.

   d) If *A* indicates IRD_HANDLE, then Value is set to the handle of the implementation row descriptor associated with *S*.

   e) If *A* indicates CURSOR SCROLLABLE, then

      Case:

      i) If the implementation supports scrollable cursors, then

         Case:

         1) If the value of the CURSOR SCROLLABLE attribute of *S* is NONSCROLLABLE, then Value is set to the code value for NONSCROLLABLE from Table 22, "Miscellaneous codes used in CLI".

         2) If the value of the CURSOR SCROLLABLE attribute of *S* is SCROLLABLE, then Value is set to the code value for SCROLLABLE from Table 22, "Miscellaneous codes used in CLI".

ii) Otherwise, an exception condition is raised: *CLI-specific condition — optional feature not implemented*.

f) If *A* indicates CURSOR SENSITIVITY, then

Case:

i) If the implementation supports cursor sensitivity, then

Case:

1) If the value of the CURSOR SENSITIVITY attribute of *S* is UNSPECIFIED, then Value is set to the code value for UNSPECIFIED from Table 22, "Miscellaneous codes used in CLI".

2) If the value of the CURSOR SENSITIVITY attribute of *S* is INSENSITIVE, then Value is set to the code value for INSENSITIVE from Table 22, "Miscellaneous codes used in CLI".

ii) Otherwise, an exception condition is raised: *CLI-specific condition — optional feature not implemented*.

# 6.36 GetTypeInfo

## Function

Get information about one or all supported data types.

## Definition

```
GetTypeInfo (
       StatementHandle          IN       INTEGER,
       DataType                 IN       SMALLINT )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If an open cursor is associated with *S*, then an exception condition is raised: *invalid cursor state*.

3) Let *D* be the value of DataType.

4) If *D* does not indicate ALL TYPES and is not one of the code values in Table 36, "Codes used for concise data types", then an exception condition is raised: *CLI-specific condition — invalid data type*.

5) Let *C* be the allocated SQL-connection with which *S* is associated.

6) Let *EC* be the established SQL-connection associated with *C* and let *SS* be the SQL-server associated with *EC*.

7) Let *TYPE_INFO* be a table, with a definition and description as specified below, that contains a row for each data type supported by *SS*. For all supported data types for which more than one name is supported, it is implementation-defined whether *TYPE_INFO* contains a single row or a row for each supported name.

```
CREATE TABLE TYPE_INFO
       (
         TYPE_NAME            CHARACTER VARYING(128) NOT NULL
           PRIMARY KEY,
         DATA_TYPE            SMALLINT               NOT NULL,
         COLUMN_SIZE          INTEGER,
         LITERAL_PREFIX       CHARACTER VARYING(128),
         LITERAL_SUFFIX       CHARACTER VARYING(128),
         CREATE_PARAMS        CHARACTER VARYING(128)
           CHARACTER SET SQL_TEXT,
         NULLABLE             SMALLINT               NOT NULL
           CHECK ( NULLABLE IN (0, 1, 2) ),
         CASE_SENSITIVE       SMALLINT               NOT NULL
           CHECK ( CASE_SENSITIVE IN (0, 1) ),
         SEARCHABLE           SMALLINT               NOT NULL
           CHECK ( SEARCHABLE IN (0, 1, 2, 3) ),
         UNSIGNED_ATTRIBUTE   SMALLINT
           CHECK ( UNSIGNED_ATTRIBUTE IN (0, 1)
                 OR UNSIGNED_ATTRIBUTE IS NULL),
         FIXED_PREC_SCALE     SMALLINT NOT NULL
           CHECK ( FIXED_PREC_SCALE IN (0, 1),
         AUTO_UNIQUE_VALUE    SMALLINT  NOT NULL
```

```
    CHECK ( AUTO_UNIQUE_VALUE IN (0, 1),
  LOCAL_TYPE_NAME     CHARACTER VARYING(128)
    CHARACTER SET SQL_TEXT,
  MINIMUM_SCALE       INTEGER,
  MAXIMUM_SCALE       INTEGER,
  SQL_DATA_TYPE       SMALLINT                NOT NULL,
  SQL_DATETIME_SUB    SMALLINT
    CHECK ( SQL_DATETIME_SUB IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)
          OR SQL_DATETIME_SUB IS NULL),
  NUM_PREC_RADIX      INTEGER,
  INTERVAL_PRECISION  SMALLINT
)
```

8) The description of the table TYPE_INFO is:

a) The value of TYPE_NAME is the name of the data type. If multiple names are supported for this data type and TYPE_INFO contains only a single row for this data type, then it is implementation-defined which of the names is in TYPE_NAME.

b) The value of DATA_TYPE is the code value for the data type as defined in Table 36, "Codes used for concise data types".

c) The value of COLUMN_SIZE is:

i) The null value if the data type has neither a length nor a precision.

ii) The maximum length in characters for a character string data type.

iii) The maximum length in bits for a bit string data type.

iv) The maximum or fixed precision, as appropriate, for a numeric data type.

v) The maximum or fixed length in positions, as appropriate, for a datetime or interval data type.

vi) An implementation-defined value for an implementation-defined data type that has a length or a precision.

d) The value of LITERAL_PREFIX is the character string that must precede the data type value when a <literal> of this data type is specified. The value of LITERAL_PREFIX is the null value if no such string is required.

e) The value of LITERAL_SUFFIX is the character string that must follow the data type value when a <literal> of this data type is specified. The value of LITERAL_SUFFIX is the null value if no such string is required.

f) The value of CREATE_PARAMS is a comma-separated list of specifiable attributes for the data type in the order in which the attributes may be specified. The attributes <length>, <precision>, <scale>, and <time fractional seconds precision> appear in the list as LENGTH, PRECISION, SCALE, and PRECISION, respectively. The appearance of attributes in implementation-defined data types is implementation-defined.

g) The value of NULLABLE is 1.

h) The value of CASE_SENSITIVE is 1 if the data type is a character string type and the default collation for its implementation-defined implicit character repertoire would result in a case sensitive comparison when two values with this data type are compared. Otherwise, the value of CASE_SENSITIVE is 0.

i) Refer to the <comparison predicate>, <between predicate>, <in predicate>, <null predicate>, <quantified comparison predicate>, and <match predicate> as the *basic predicates*. If the data type can be the data type of an operand in the <like predicate>, then let *V1* be 1; otherwise let *V1* be 0. If the data type can be the data type of a column of a <row value constructor> immediately contained in a basic predicate, then let *V2* be 2; otherwise let *V2* be 0. The value of SEARCHABLE is $(V1 + V2)$.

j) The value of UNSIGNED_ATTRIBUTE is

Case:

i) If the data type is unsigned, then 1.

ii) If the data type is signed, then 0.

iii) If a sign is not applicable to the data type, then the null value.

k) The value of FIXED_PREC_SCALE is

Case:

i) If the data type is an exact numeric with a fixed precision and scale, then 1.

ii) Otherwise, 0.

l) The value of AUTO_UNIQUE_VALUE is

Case:

i) If a column of this data type is set to a value unique among all rows of that column when a row is inserted, then 1.

ii) Otherwise, 0.

m) The value of LOCAL_TYPE_NAME is an implementation-defined localized representation of the name of the data type, intended primarily for display purposes. The value of LOCAL_TYPE_NAME is the null value if a localized representation is not supported.

n) The value of MINIMUM_SCALE is:

i) The null value if the data type has neither a scale nor a fractional seconds precision.

ii) The minimum value of the scale for a data type that has a scale.

iii) The minimum value of the fractional seconds precision for a data type that has a fractional seconds precision.

o) The value of MAXIMUM_SCALE is:

i) The null value if the data type has neither a scale nor a fractional seconds precision.

ii) The maximum value of the scale for a data type that has a scale.

iii) The maximum value of the fractional seconds precision for a data type that has a fractional seconds precision.

p) The value of SQL_DATA_TYPE is the code value for the data type as defined in Table 6, "Codes used for implementation data types in CLI".

q) The value of SQL_DATETIME_SUB is

Case:

i) If the data type is a datetime type, then the code value for the datetime type as defined in Table 8, "Codes associated with datetime data types in SQL/CLI".

ii) If the data type is an interval type, then the code value for the interval type as defined in Table 9, "Codes associated with <interval qualifier> in SQL/CLI".

iii) Otherwise, the null value.

r) The value of NUM_PREC_RADIX is

Case:

i) If the value of PRECISION is the value of a precision, then the radix of that precision.

ii) Otherwise, the null value.

s) The value of SQL_INTERVAL_PRECISION is

Case:

i) If the data type is an interval type, then <interval leading field precision>.

ii) Otherwise, the null value.

9) Case:

a) If *D* indicates ALL TYPES, then let *P* be the character string

```
SELECT * FROM TYPE_INFO
```

b) Otherwise, let *P* be the character string

```
SELECT * FROM TYPE_INFO WHERE DATA_TYPE = D
```

10) ExecDirect is implicitly invoked with *S* as the value of StatementHandle, *P* as the value of StatementText, and the length of *P* as the value of TextLength.

# 6.37 NumResultCols

## Function

Get the number of result columns.

## Definition

```
NumResultCols (
       StatementHandle       IN        INTEGER,
       ColumnCount           OUT       SMALLINT )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Case:

   a) If there is no prepared or executed statement associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

   b) Otherwise, let *D* be the implementation row descriptor associated with *S* and let *N* be the value of COUNT for *D*.

3) ColumnCount is set to *N*.

# 6.38 ParamData

## Function

Process a deferred parameter value.

## Definition

```
ParamData (
       StatementHandle      IN    INTEGER,
       Value                OUT   ANY )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Case:

   a) If there is no deferred parameter number associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

   b) Otherwise, let *DPN* be the deferred parameter number associated with *S*.

3) Let *APD* be the current application parameter descriptor for *S* and let *N* be the value of COUNT for *APD*.

4) For each item descriptor area for which DEFERRED is true in the first *N* item descriptor areas of *APD*, refer to the corresponding <dynamic parameter specification> value as a deferred parameter value.

5) Let *IDA* be the *DPN*-th item descriptor area of *APD* and let *PT* and *DP* be the values of TYPE and DATA_POINTER, respectively, for *IDA*.

6) If there is no parameter value associated with *DPN*, then

   Case:

   a) If there is a DATA_POINTER value associated with *DPN*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

   b) Otherwise:

      i) Value is set to *DP*.

      ii) *DP* becomes the DATA_POINTER value associated with *DPN*.

      iii) An exception condition is raised: *CLI-specific condition — dynamic parameter value needed*.

7) Let *IPD* be the implementation parameter descriptor associated with *S*.

8) Let *C* be the allocated SQL-connection with which *S* is associated.

9) Let *V* be the parameter value associated with *DPN*.

10) Case:

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

i) Case:

1) If *PT* indicates CHARACTER, then let *L* be the parameter length associated with *DPN*.

2) Otherwise, let *L* be zero.

ii) 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 for *IDA*.

b) Otherwise, let *SV* be the null value.

11) Let *TDT* be the effective data type of the *DPN*-th <dynamic parameter specification> 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 *DPN*-th item descriptor area of *IPD*.

12) 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*.

13) 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.

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

```
CAST (SV AS TDT)
```

NOTE 35 – It is implementation-dependent whether the establishment of *TV* occurs at this time or during the preceding invocation of PutData.

15) Let *ADT* be the effective data type of the actual *DPN*-th <dynamic parameter specification>, 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 *DPN*-th item descriptor area of *IPD* if POPULATE IPD was *true* for *C*.

16) 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*.

17) 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.

18) The <cast specification>

```
CAST (TV AS ADT)
```

is effectively performed and is the value of the *DPN*-th dynamic parameter.

19) Let *PN* be the parameter number associated with a deferred parameter value and let *HPN* be the value of MAX(*PN*).

20) If *DPN* is not equal to *HPN*, then:

a) Let *NPN* be the lowest value of *PN* for which $DPN < NPN <= HPN$.

b) Let *DP* be the value of DATA_POINTER for the *NPN*-th item descriptor area of *APD*.

c) *NPN* becomes the deferred parameter number associated with *S* and *DP* becomes the DATA_POINTER value associated with the deferred parameter number.

d) An exception condition is raised: *CLI-specific condition — dynamic parameter value needed*.

21) If *DPN* is equal to *HPN*, then:

a) *DPN* is removed from association with *S*.

b) Case:

i) If there is a select source associated with *S*, then:

1) Let *SS* be the select source associated with *S*.

2) If the value of the CURSOR SCROLLABLE attribute of *S* is SCROLLABLE, then let *CT* be 'SCROLL'; otherwise, let *CT* be an empty string.

3) If the value of the CURSOR SENSITIVITY attribute of *S* is INSENSITIVE, then let *CS* be 'INSENSITIVE'; otherwise, let *CS* be an empty string.

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

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

5) A copy of *SS* is effectively created in which:

A) Each <dynamic parameter specification> is replaced by the value of the corresponding dynamic parameter.

B) 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 time; and

C) Each <datetime value function> generally contained in *SS* is replaced by the value resulting from evaluation of that <datetime value function>, with all evaluations effectively done at the same time.

6) Let *T* be the table specified by the copy of *SS*.

7) A table descriptor for *T* is effectively created.

8) The General Rules of Subclause 13.1, "<declare cursor>", in ISO/IEC 9075:1992, are applied to *CR*.

9) Case:

A) If *CR* specified INSENSITIVE, then a copy of *T* is effectively created and the cursor identified by *CN* is placed in the open state and its position is before the first row of the copy of *T*.

B) Otherwise, the cursor identified by *CN* is placed in the open state and its position is before the first row of *T*.

ii) Otherwise:

1) Let *SS* be the statement source associated with *S*.

2) *SS* is removed from association with *S*.

3) Case:

A) If *SS* is a <preparable dynamic delete statement: positioned>, then:

I) Let *CR* be the cursor referenced by *SS*.

II) All the General Rules in Subclause 17.19, "<preparable dynamic delete statement: positioned>", in ISO/IEC 9075:1992, apply to *SS*.

III) If the execution of *SS* deleted the current row of *CR*, then the effect on the fetched row, if any, associated with the allocated SQL-statement under which that current row was established, is implementation-defined.

B) If *SS* is a <preparable dynamic update statement: positioned>, then:

I) Let *CR* be the cursor referenced by *SS*.

II) All the General Rules in Subclause 17.20, "<preparable dynamic update statement: positioned>", in ISO/IEC 9075:1992, apply to *SS*.

III) If the execution of *SS* updated the current row of *CR*, then the effect on the fetched row, if any, associated with the allocated SQL-statement under which that current row was established, is implementation-defined.

C) Otherwise, the results of the execution are the same as if the statement were contained in a <procedure> and executed; these are described in Subclause 12.4, "Calls to a <procedure>", in ISO/IEC 9075:1992.

c) Let *R* be the value of the ROW_COUNT field from the diagnostics area associated with *S*.

d) *R* becomes the row count associated with *S*.

e) If *P* executed successfully, then any executed statement associated with *S* is destroyed and *SS* becomes the executed statement associated with *S*.

# 6.39 Prepare

## Function

Prepare a statement.

## Definition

```
Prepare (
       StatementHandle       IN        INTEGER,
       StatementText         IN        CHARACTER(L),
       TextLength            IN        SMALLINT )
       RETURNS SMALLINT
```

where *L* is determined by the value of TextLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If an open cursor is associated with *S*, then an exception condition is raised: *invalid cursor state*.

3) Let *TL* be the value of TextLength.

4) Case:

   a) If *TL* is not negative, then let *L* be *TL*.

   b) If *TL* indicates NULL TERMINATED, then let *L* be the number of octets of StatementText that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *P* be the first *L* octets of StatementText.

6) If *P* is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then let *CN* be the cursor name referenced by *P*. Let *C* be the allocated SQL-connection with which *S* is associated. If *CN* is not the name of a cursor associated with another allocated SQL-statement associated with *C*, then an exception condition is raised: *invalid cursor name*.

7) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation*.

   a) *P* does not conform to the Format, Syntax Rules or Access Rules for a <preparable statement> or *P* is a <commit statement> or a <rollback statement>.

NOTE 36 – See Table 11, "SQL-statement character codes for use in a diagnostics area", for the list of <preparable statement>s.

b) *P* contains a <comment>.

c) *P* contains a <dynamic parameter specification> in an invalid position as determined by the rules specified in Subclause 17.6, "<prepare statement>", in ISO/IEC 9075:1992.

8) The data type of any <dynamic parameter specification> contained in *P* is determined by the rules specified in Subclause 17.6, "<prepare statement>", in ISO/IEC 9075:1992.

9) The following objects associated with *S* are destroyed:

a) Any prepared statement.

b) Any cursor.

c) Any select source.

d) Any executed statement.

If a cursor associated with *S* is destroyed, then so are any prepared statements that reference that cursor.

10) *P* is prepared and the prepared statement is associated with *S*.

11) If *P* is a <dynamic select statement> or a <dynamic single row select statement>, then:

a) *P* becomes the select source associated with *S*.

b) If there is no cursor name associated with *S*, then a unique implementation-dependent name that has the prefix 'SQLCUR' or the prefix 'SQL_CUR' becomes the cursor name associated with *S*.

12) The General Rules of Subclause 5.3.3, "Implicit using clause", are applied to 'DESCRIBE', *P*, and *S* as *TYPE*, *SOURCE*, and *ALLOCATED STATEMENT*, respectively.

13) The validity of a prepared statement in an SQL-transaction different from the one in which the statement was prepared is implementation-dependent.

# 6.40 PutData

## Function

Provide a deferred parameter value.

## Definition

```
PutData (
       StatementHandle      IN    INTEGER,
       Data                 IN    ANY,
       StrLen_or_Ind        IN    INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Case:

   a) If there is no deferred parameter number associated with *S*, then an exception condition is raised: *CLI-specific condition — function sequence error*.

   b) Otherwise, let *DPN* be the deferred parameter number associated with *S*.

3) If there is no DATA_POINTER value associated with DPN, then an exception condition is raised: *CLI-specific condition — function sequence error*.

4) Let *APD* be the current application parameter descriptor for *S*.

5) Let *PT* be the value of TYPE for the *DPN*-th item descriptor area of *APD*.

6) Let *IV* be the value of StrLen_or_Ind.

7) If there is a parameter value associated with *DPN* and *PT* does not indicate CHARACTER, then an exception is raised: *CLI-specific condition — non-string data cannot be sent in pieces*.

8) Case:

   a) If *IV* is −1, then let *V* be the null value.

   b) If *PT* indicates CHARACTER, then:

      i) Case:

         1) If *IV* is not negative, then let *L* be the number of whole characters in the first *IV* octets of *V*.

         2) If *IV* indicates NULL TERMINATED, then let *L* be the number of characters of Data that precede the implementation-defined null character that terminates a C character string.

         3) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

ii) Case:

1) If $L$ is not a valid length value for a CHARACTER data type, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

2) Otherwise, let $V$ be the first $L$ characters of Data.

c) Otherwise, let $V$ be the value of Data.

9) If $V$ is not a valid value of the data type indicated by $PT$, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

10) If there is no parameter value associated with *DPN*, then:

a) $V$ becomes the parameter value associated with *DPN*.

b) If $V$ is not the null value and $PT$ indicates CHARACTER, then $L$ becomes the parameter length associated with *DPN*.

11) If there is a parameter value associated with *DPN*, then

Case:

a) If $V$ is the null value, then:

i) *DPN* is removed from association with $S$.

ii) Any statement source associated with $S$ is removed from association with $S$.

iii) An exception condition is raised: *CLI-specific condition — attempt to concatenate a null value*.

b) Otherwise:

i) Let *PV* be the parameter value associated with *DPN*.

ii) Case:

1) If *PV* is the null value, then:

A) *DPN* is removed from association with $S$.

B) Any statement source associated with $S$ is removed from association with $S$.

C) An exception condition is raised: *CLI-specific condition — attempt to concatenate a null value*.

2) Otherwise:

A) Let *PL* be the parameter length associated with *DPN*.

B) Let *NV* be the result of the <string value function>

```
PV || V
```

C) *NV* becomes the parameter value associated with *DPN* and $(PL+L)$ becomes the parameter length associated with *DPN*.

# 6.41 RowCount

## Function

Get the row count.

## Definition

```
RowCount (
       StatementHandle      IN          INTEGER )
       RowCount             OUT         INTEGER )
       RETURNS SMALLINT
```

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If there is no executed statement associated with *S*, then an exception condition is raised: *CLI-specific conditon — function sequence error*.

3) RowCount is set to the value of the row count associated with *S*.

# 6.42 SetConnectAttr

## Function

Set the value of an SQL-connection attribute.

## Definition

```
SetConnectAttr(
      ConnectionHandle     IN    INTEGER,
      Attribute            IN    INTEGER,
      Value                IN    ANY,
      StringLength         IN    INTEGER )
      RETURNS SMALLINT
```

## General Rules

1) Case:

   a) If ConnectionHandle does not identify an allocated SQL-connection, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *C* be the allocated SQL-connection identified by ConnectionHandle.

      ii) The diagnostics area associated with *C* is emptied.

2) Let *A* be the value of Attribute.

3) If *A* is not one of the code values in Table 16, "Codes used for connection attributes", or if *A* is one of the code values in Table 16, "Codes used for connection attributes", but the row that contains *A* contains "No" in the "May be set" column, then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

# 6.43 SetCursorName

## Function

Set a cursor name.

## Definition

```
SetCursorName (
       StatementHandle      IN    INTEGER,
       CursorName           IN    CHARACTER(L),
       NameLength           IN    SMALLINT )
       RETURNS SMALLINT
```

where *L* is determined by the value of NameLength and has a maximum value equal to the implementation-defined maximum length of a variable-length character string.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) If an open cursor is associated with *S*, then an exception condition is raised: *invalid cursor state*.

3) Let *NL* be the value of NameLength.

4) Case:

   a) If *NL* is not negative, then let *L* be *NL*.

   b) If *NL* indicates NULL TERMINATED, then let *L* be the number of octets of CursorName that precede the implementation-defined null character that terminates a C character string.

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

5) Case:

   a) If *L* is zero, then an exception condition is raised: *CLI-specific condition — invalid string length or buffer length*.

   b) Otherwise, let *CV* be the first *L* octets of CursorName and let *TCN* be the value of

      ```
      TRIM ( BOTH ' ' FROM CV )
      ```

6) Let *ML* be the maximum length in characters allowed for an <identifier> as specified in the Syntax Rules of Subclause 5.4, "Names and identifiers", in ISO/IEC 9075:1992, and let *TCNL* be the length in characters of *TCN*.

7) Case:

   a) If *TCNL* is greater than *ML*, then *CN* is set to the first *ML* characters of *TCN* and a completion condition is raised: *warning — string data, right truncation*.

   b) Otherwise, *CN* is set to *TCN*.

8) If *CN* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid cursor name*.

NOTE 37 – An <identifier> may optionally contain "<introducer><character set name>". See ISO/IEC 9075:1992, Subclause 5.4, "Names and identifiers".

9) Let *C* be the allocated SQL-connection with which *S* is associated and let *SC* be the <search condition>:

```
CN LIKE 'SQL\_CUR%' ESCAPE '\' OR CN LIKE 'SQLCUR%'
```

If *SC* is true or if *CN* is identical to the value of any cursor name associated with an allocated SQL-statement associated with *C*, then an exception condition is raised: *invalid cursor name*.

10) *CN* becomes the cursor name associated with *S*.

# 6.44 SetDescField

## Function

Set a field in a CLI descriptor area.

## Definition

```
SetDescField (
      DescriptorHandle      IN      INTEGER,
      RecordNumber          IN      SMALLINT,
      FieldIdentifier       IN      SMALLINT,
      Value                 IN      ANY,
      BufferLength          IN      INTEGER )
      RETURNS SMALLINT
```

NOTE 38 – The BufferLength parameter is not used at present.

## General Rules

1) Let *D* be the allocated CLI descriptor area identified by DescriptorHandle.

2) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *D* as the DESCRIPTOR AREA.

3) If *D* is an implementation row descriptor, then an exception condition is raised: *CLI-specific condition — cannot modify an implementation row descriptor*.

4) Let *FI* be the value of FieldIdentifier.

5) If *FI* is not one of the code values in Table 20, "Codes used for descriptor fields", or if it is one of the code values in Table 20, "Codes used for descriptor fields", but does not indicate COUNT, TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, OCTET_LENGTH_POINTER, DATA_POINTER, or INDICATOR_POINTER, then an exception condition is raised: *CLI-specific condition — invalid descriptor field identifier*.

6) Let *RN* be the value of RecordNumber.

7) If *FI* does not indicate COUNT and *RN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

8) Let *IDA* be the item descriptor area of *D* specified by *RN*.

9) Information is set in *D*:

   Case:

   a) If *FI* indicates COUNT, then the count of the number of item descriptor areas is set to the value of Value.

   b) If *FI* indicates LENGTH_POINTER, then the value of the LENGTH_POINTER field of *IDA* is set to the address of Value.

c) If *FI* indicates DATA_POINTER, then the value of the DATA_POINTER field of *IDA* is set to the address of Value. If Value is a null pointer, then the address is set to 0.

d) If *FI* indicates INDICATOR_POINTER, then the value of the INDICATOR_POINTER field of *IDA* is set to the address of Value.

e) Otherwise, the value of the field of *IDA* identified by *FI* is set to the value of Value.

10) If *FI* indicates TYPE, LENGTH, OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISON, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, or CHARACTER_SET_NAME, then DATA_POINTER for *IDA* is set to zero.

11) If *FI* indicates DATA_POINTER, and Value is not a null pointer, and *IDA* is not consistent as specified in Subclause 5.3.8, "Description of CLI item descriptor areas", then an exception condition is raised: *CLI-specific condition — inconsistent descriptor information*.

12) Let *V* be the value of Value.

13) If *FI* indicates TYPE, then:

a) All the other fields of *IDA* are set to implementation-dependent values.

b) Case:

i) If *V* indicates CHARACTER or CHARACTER VARYING, then the CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME fields of *IDA* are set to the values for the default character set name for the SQL-session and the LENGTH field of *IDA* is set to the maximum possible length in characters of the indicated data type.

ii) If *V* indicates BIT or BIT VARYING, then the LENGTH field of *IDA* is set to the maximum possible length in bits of the indicated data type.

iii) If *V* indicates a <datetime type>, then the PRECISION field of *IDA* is set to 0.

iv) If *V* indicates INTERVAL, then the DATETIME_INTERVAL_PRECISION field of *IDA* is set to 2.

v) If *V* indicates NUMERIC or DECIMAL, then the SCALE field of *IDA* is set to 0 and the PRECISION field of *IDA* is set to the implementation-defined default value for the precision of NUMERIC or DECIMAL data types, respectively.

vi) If *V* indicates FLOAT, then the PRECISION field of *IDA* is set to the implementation-defined default value for the precision of FLOAT data types.

14) If *FI* indicates DATETIME_INTERVAL_CODE and the TYPE field of *IDA* indicates a <datetime type>, then:

a) All the fields of *IDA* other than DATETIME_INTERVAL_CODE and TYPE are set to implementation-dependent values.

b) Case:

i) If *V* indicates DATE, TIME, or TIME WITH TIME ZONE, then the PRECISION field of *IDA* is set to 0.

ii) If *V* indicates TIMESTAMP or TIMESTAMP WITH TIME ZONE, then the PRECISION field of *IDA* is set to 6.

15) If *FI* indicates DATETIME_INTERVAL_CODE and the TYPE field of *IDA* indicates INTERVAL, then the DATETIME_INTERVAL_PRECISION field of IDA is set to 2 and

a) If *V* indicates DAY TO SECOND, HOUR TO SECOND, MINUTE TO SECOND, or SECOND, then the PRECISION field of *IDA* is set to 6.

b) Otherwise, the PRECISION field of *IDA* is set to 0.

16) If an exception condition is raised, then the field of *IDA* indicated by *FI* is set to an implementation-dependent value.

17) Restrictions on the differences allowed between corresponding implementation and application descriptors are implementation-defined, except as specified in the General Rules of Subclause 5.3.3, "Implicit using clause", in the General Rules of Subclause 6.27, "GetData", and in the General Rules of Subclause 6.38, "ParamData".

# 6.45 SetDescRec

## Function

Set commonly-used fields in a CLI descriptor area.

## Definition

```
SetDescRec (
       DescriptorHandle      IN        INTEGER,
       RecordNumber          IN        SMALLINT,
       Type                  IN        SMALLINT,
       SubType               IN        SMALLINT,
       Length                IN        INTEGER,
       Precision             IN        SMALLINT,
       Scale                 IN        SMALLINT,
       Data                  DEF       ANY,
       StringLength          DEF       INTEGER,
       Indicator             DEF       SMALLINT )
       RETURNS SMALLINT
```

## General Rules

1) Let *D* be the allocated CLI descriptor area identified by DescriptorHandle and let *N* be the value of COUNT for *D*.

2) The General Rules of Subclause 5.3.5, "Deferred parameter check", are applied to *D* as the DESCRIPTOR AREA.

3) If *D* is an implementation row descriptor, then an exception condition is raised: *CLI-specific condition — cannot modify an implementation row descriptor*.

4) Let *RN* be the value of RecordNumber.

5) If *RN* is less than 1, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

6) Let *IDA* be the item descriptor area of *D* specified by *RN*.

7) Information is set in *D* as follows:

   a) The data type, precision, scale, and datetime data type of the item described by *IDA* are set to the values of Type, Precision, Scale, and SubType, respectively.

   b) Case:

      i) If *D* is an implementation parameter descriptor, then the length (in characters, bits, or positions, as appropriate) of the item described by *IDA* is set to the value of Length.

      ii) Otherwise, the length in octets of the item described by *IDA* is set to the value of Length.

   c) If StringLength is not a null pointer, then the address of the host variable that is to provide the length of the item described by *IDA*, or that is to receive the returned length in octets of the item described by *IDA*, is set to the address of StringLength.

d) The address of the host variable that is to provide a value for the item described by *IDA*, or that is to receive a value for the item described by *IDA*, is set to the address of Data. If Data is a null pointer, then the address is set to 0.

e) If Indicator is not a null pointer, then the address of the <indicator variable> associated with the item described by *IDA* is set to the address of Indicator.

8) If Data is not a null pointer *IDA* is not consistent as specified in Subclause 5.3.8, "Description of CLI item descriptor areas", then an exception condition is raised: *CLI-specific condition — inconsistent descriptor information*.

9) If *RN* is greater than *N*, then the COUNT field of *D* is set to *RN*.

10) If an exception condition is raised, then all fields of *IDA* for which specific values were provided in the invocation of SetDescRec are set to implementation-dependent values and the value of COUNT for *D* is unchanged.

11) Restrictions on the differences allowed between corresponding implementation and application descriptors are implementation-defined, except as specified in the General Rules of Subclause 5.3.3, "Implicit using clause", in the General Rules of Subclause 6.27, "GetData", and in the General Rules of Subclause 6.38, "ParamData".

# 6.46 SetEnvAttr

## Function

Set the value of an SQL-environment attribute.

## Definition

```
SetEnvAttr (
      EnvironmentHandle     IN        INTEGER,
      Attribute             IN        INTEGER,
      Value                 IN        ANY,
      StringLength          IN        INTEGER )
      RETURNS SMALLINT
```

NOTE 39 – The StringLength parameter is not used at present.

## General Rules

1) Case:

   a) If EnvironmentHandle does not identify an allocated SQL-environment or if it identifies an allocated skeleton SQL-environment, then an exception condition is raised: *CLI-specific condition — invalid handle*.

   b) Otherwise:

      i) Let *E* be the allocated SQL-environment identified by EnvironmentHandle.

      ii) The diagnostics area associated with *E* is emptied.

2) If there are any allocated SQL-connections associated with *E*, then an exception condition is raised: *CLI-specific condition — attribute cannot be set now*.

3) Let *A* be the value of Attribute.

4) If *A* is not one of the code values in Table 15, "Codes used for environment attributes", then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

5) If *A* indicates NULL TERMINATION, then

   Case:

   a) If Value indicates TRUE, then null termination for *E* is set to *true* .

   b) If Value indicates FALSE, then null termination for *E* is set to *false* .

   c) Otherwise, an exception condition is raised: *CLI-specific condition — invalid attribute value*.

# 6.47 SetStmtAttr

## Function

Set the value of an SQL-statement attribute.

## Definition

```
SetStmtAttr (
       StatementHandle       IN        INTEGER,
       Attribute             IN        INTEGER,
       Value                 IN        ANY,
       StringLength          IN        INTEGER )
       RETURNS SMALLINT
```

NOTE 40 – The StringLength parameter is not used at present.

## General Rules

1) Let *S* be the allocated SQL-statement identified by StatementHandle.

2) Let *A* be the value of Attribute.

3) If *A* is not one of the code values in Table 17, "Codes used for statement attributes", or if *A* is one of the code values in Table 17, "Codes used for statement attributes", but the row that contains *A* contains "No" in the "May be set" column, then an exception condition is raised: *CLI-specific condition — invalid attribute identifier*.

4) Let *V* be the value of Value.

5) Case:

   a) If *A* indicates APD_HANDLE, then:

      i) Case:

         1) If *V* does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid attribute value*.

         2) Otherwise, let *DA* be the CLI descriptor area identified by *V* and let *AT* be the value of the ALLOC_TYPE field for *DA*.

      ii) Case:

         1) If *AT* indicates AUTOMATIC but *DA* is not the application parameter descriptor associated with *S*, then an exception condition is raised: *CLI-specific condition — invalid use of automatically-allocated descriptor handle*.

         2) Otherwise, *DA* becomes the current application parameter descriptor for *S*.

   b) If *A* indicates ARD_HANDLE, then:

      i) Case:

         1) If *V* does not identify an allocated CLI descriptor area, then an exception condition is raised: *CLI-specific condition — invalid attribute value*.

2) Otherwise, let *DA* be the CLI descriptor area identified by *V* and let *AT* be the value of the ALLOC_TYPE field for *DA*.

ii) Case:

1) If *AT* indicates AUTOMATIC but *DA* is not the application row descriptor associated with *S*, then an exception condition is raised: *CLI-specific condition — invalid use of automatically-allocated descriptor handle*.

2) Otherwise, *DA* becomes the current application row descriptor for *S*.

c) If *A* indicates CURSOR SCROLLABLE, then

Case:

i) If the implementation supports scrollable cursors, then

Case:

1) If an open cursor is associated with *S*, then an exception condition is raised: *CLI-specific condition — attribute cannot be set now*.

2) Case:

A) If *V* indicates NONSCROLLABLE, then the CURSOR SCROLLABLE attribute of *S* is set to NONSCROLLABLE.

B) If *V* indicates SCROLLABLE, then the CURSOR SCROLLABLE attribute of *S* is set to SCROLLABLE.

C) Otherwise, an exception condition is raised: *CLI-specific condition — invalid attribute value*.

ii) Otherwise, an exception condition is raised: *CLI-specific condition — optional feature not implemented*.

d) If *A* indicates CURSOR SENSITIVITY, then

Case:

i) If the implementation supports cursor sensitivity, then

Case:

1) If an open cursor is associated with *S*, then an exception condition is raised: *CLI-specific condition — attribute cannot be set now*.

2) Case:

A) If *V* indicates UNSPECIFIED, then the CURSOR SENSITIVITY attribute of *S* is set to UNSPECIFIED.

B) If *V* indicates INSENSITIVE, then the CURSOR SENSITIVITY attribute of *S* is set to INSENSITIVE.

C) Otherwise, an exception condition is raised: *CLI-specific condition — invalid attribute value*.

ii) Otherwise, an exception condition is raised: *CLI-specific condition — optional feature not implemented*.

# 7 Conformance

## 7.1 Introduction

This part of ISO/IEC 9075 specifies conforming SQL/CLI routines and conforming SQL/CLI implementations.

A conforming SQL/CLI routine shall abide by the BNF Format and associated Syntax Rules, Access Rules, Definitions, and General Rules specified for <CLI routine>s in this part of ISO/IEC 9075 and for a stated *level of conformance* for <preparable statement>s in ISO/IEC 9075:1992.

A conforming SQL/CLI implementation shall process conforming SQL/CLI routines according to the associated Definitions and General Rules specified for <CLI routine>s in this part of ISO/IEC 9075, and for a stated *level of conformance* for <preparable statement>s in ISO/IEC 9075:1992.

## 7.2 Claims of conformance

Claims of conformance to this part of ISO/IEC 9075 shall state:

1) Which *level of conformance* is claimed for <preparable statement>s. SQL levels of conformance are specified in Subclause 23.2, "Claims of conformance", of ISO/IEC 9075:1992. All aspects of this part of ISO/IEC 9075 shall be implemented without regard to the claimed level of conformance other than:

   a) An implementation need not support the automatic population of the Implementation Parameter Descriptor (IPD) unless that implementation supports Full SQL. An implementation that does not support the automatic population of the IPD returns a “false” value when the GetConnectAttr routine is called with POPULATE IPD attribute.

   b) An implementation need not support scrollable cursors unless that implementation supports Intermediate SQL. An implementation that does not support scrollable cursors does not allow the SetStmtAttr routine to set the CURSOR SCROLLABLE attribute to SCROLLABLE.

   c) An implementation need not support insensitive cursors unless that implementation supports Full SQL. An implementation that does not support insensitive cursors does not allow the SetStmtAttr routine to set the CURSOR SENSITIVITY attribute to INSENSITIVE.

2) Which of the following standard programming languages are supported for SQL/CLI routine invocation (see Subclause 5.2, “<CLI routine> invocation”):

   a) Ada

   b) C

   c) COBOL

   d) Fortran

   e) MUMPS

f) Pascal

g) PL/I

3) The definitions for all elements and actions that this part of ISO/IEC 9075 and the associated clauses of ISO/IEC 9075:1992 specify as implementation-defined.

## 7.3 Extensions and options

A conforming implementation may provide support for additional implementation-defined routines or for implementation-defined argument values for <CLI routine>s.

An implementation remains conforming even if it provides user options to process conforming <CLI routine> invocations in a nonconforming manner.

# Annex A
(informative)

# Typical header files

## A.1 C Header File SQLCLI.H

Here is a typical SQLCLI.H file. C applications include this file by containing the following statement:

```
#include "sqlcli.h"
```

The following file contains C language function prototypes for the SQL/CLI routines.

```
/* sqlcli.h  Header File for SQL CLI.
 * The actual header file must contain at least the information
 * specified here, except that the comments may vary.
 */

/* API declaration data types */
typedef unsigned char   SQLCHAR;
typedef long            SQLINTEGER;
typedef short           SQLSMALLINT;
typedef double          SQLDOUBLE;
typedef float           SQLREAL;
typedef void *          SQLPOINTER;
typedef unsigned char   SQLDATE;
typedef unsigned char   SQLTIME;
typedef unsigned char   SQLTIMESTAMP;
typedef unsigned char   SQLDECIMAL;
typedef unsigned char   SQLNUMERIC;

/* function return type */
typedef SQLSMALLINT     SQLRETURN;

/* generic data structures */
typedef SQLINTEGER      SQLHENV;    /* environment handle */
typedef SQLINTEGER      SQLHDBC;    /* connection handle */
typedef SQLINTEGER      SQLHSTMT;   /* statement handle */
typedef SQLINTEGER      SQLHDESC;   /* descriptor handle */

/* special length/indicator values */
#define SQL_NULL_DATA                           -1
#define SQL_DATA_AT_EXEC                        -2

/* return values from functions */
```

```
#define SQL_SUCCESS                                0
#define SQL_SUCCESS_WITH_INFO                      1
#define SQL_NEED_DATA                             99
#define SQL_NO_DATA                              100
#define SQL_ERROR                                 -1
#define SQL_INVALID_HANDLE                        -2

/* test for SQL_SUCCESS or SQL_SUCCESS_WITH_INFO */
#define SQL_SUCCEEDED(rc) (((rc)&(~1))==0)

/* flags for null-terminated string */
#define SQL_NTS                                   -3
#define SQL_NTSL                                  -3L

/* maximum message length */
#define SQL_MAXIMUM_MESSAGE_LENGTH               512

/* maximum identifier length */
#define SQL_MAXIMUM_ID_LENGTH                     18

/* date/time length constants */
/* add p+1 for time and timestamp if precision is nonzero */
#define SQL_DATE_LENGTH                           10
#define SQL_TIME_LENGTH                            8
#define SQL_TIMESTAMP_LENGTH                      19

/* handle type identifiers */
#define SQL_HANDLE_ENV                             1
#define SQL_HANDLE_DBC                             2
#define SQL_HANDLE_STMT                            3
#define SQL_HANDLE_DESC                            4

/* environment attribute */
#define SQL_ATTR_OUTPUT_NTS                    10001

/* connection attribute */
#define SQL_ATTR_AUTO_IPD                      10001

/* statement attributes */
#define SQL_ATTR_APP_ROW_DESC                  10010
#define SQL_ATTR_APP_PARAM_DESC                10011
#define SQL_ATTR_IMP_ROW_DESC                  10012
#define SQL_ATTR_IMP_PARAM_DESC                10013
#define SQL_ATTR_CURSOR_SCROLLABLE                -1
#define SQL_ATTR_CURSOR_SENSITIVITY               -2

/* identifiers of fields in the SQL descriptor */
#define SQL_DESC_COUNT                          1001
#define SQL_DESC_TYPE                           1002
#define SQL_DESC_LENGTH                         1003
#define SQL_DESC_OCTET_LENGTH_POINTER           1004
#define SQL_DESC_PRECISION                      1005
#define SQL_DESC_SCALE                          1006
#define SQL_DESC_DATETIME_INTERVAL_CODE         1007
#define SQL_DESC_NULLABLE                       1008
#define SQL_DESC_INDICATOR_POINTER              1009
#define SQL_DESC_DATA_POINTER                   1010
#define SQL_DESC_NAME                           1011
#define SQL_DESC_UNNAMED                        1012
#define SQL_DESC_OCTET_LENGTH                   1013
#define SQL_DESC_DATETIME_INTERVAL_PRECISION    1014
#define SQL_DESC_COLLATION_CATALOG              1015
```

```
#define SQL_DESC_COLLATION_SCHEMA                1016
#define SQL_DESC_COLLATION_NAME                  1017
#define SQL_DESC_CHARACTER_SET_CATALOG           1018
#define SQL_DESC_CHARACTER_SET_SCHEMA            1019
#define SQL_DESC_CHARACTER_SET_NAME              1020
#define SQL_DESC_PARAMETER_MODE                  1021
#define SQL_DESC_PARAMETER_ORDINAL_POSITION      1022
#define SQL_DESC_PARAMETER_SPECIFIC_CATALOG      1023
#define SQL_DESC_PARAMETER_SPECIFIC_SCHEMA       1024
#define SQL_DESC_PARAMETER_SPECIFIC_NAME         1025
#define SQL_DESC_ALLOC_TYPE                      1099

/* identifiers of fields in the diagnostics area */
#define SQL_DIAG_RETURNCODE                         1
#define SQL_DIAG_NUMBER                             2
#define SQL_DIAG_ROW_COUNT                          3
#define SQL_DIAG_SQLSTATE                           4
#define SQL_DIAG_NATIVE                             5
#define SQL_DIAG_MESSAGE_TEXT                       6
#define SQL_DIAG_DYNAMIC_FUNCTION                   7
#define SQL_DIAG_CLASS_ORIGIN                       8
#define SQL_DIAG_SUBCLASS_ORIGIN                    9
#define SQL_DIAG_CONNECTION_NAME                   10
#define SQL_DIAG_SERVER_NAME                       11
#define SQL_DIAG_DYNAMIC_FUNCTION_CODE             12
#define SQL_DIAG_MORE                              13
#define SQL_DIAG_CONDITION_NUMBER                  14
#define SQL_DIAG_CONSTRAINT_CATALOG                15
#define SQL_DIAG_CONSTRAINT_SCHEMA                 16
#define SQL_DIAG_CONSTRAINT_NAME                   17
#define SQL_DIAG_CATALOG_NAME                      18
#define SQL_DIAG_SCHEMA_NAME                       19
#define SQL_DIAG_TABLE_NAME                        20
#define SQL_DIAG_COLUMN_NAME                       21
#define SQL_DIAG_CURSOR_NAME                       22
#define SQL_DIAG_MESSAGE_LENGTH                    23
#define SQL_DIAG_MESSAGE_OCTET_LENGTH              24

/* dynamic function codes returned in diagnostics area*/
#define SQL_DIAG_ALTER_DOMAIN                       3
#define SQL_DIAG_ALTER_TABLE                        4
#define SQL_DIAG_CALL                               7
#define SQL_DIAG_CREATE_ASSERTION                   6
#define SQL_DIAG_CREATE_CHARACTER_SET               8
#define SQL_DIAG_CREATE_COLLATION                  10
#define SQL_DIAG_CREATE_DOMAIN                     23
#define SQL_DIAG_CREATE_SCHEMA                     64
#define SQL_DIAG_CREATE_TABLE                      77
#define SQL_DIAG_CREATE_TRANSLATION                79
#define SQL_DIAG_CREATE_VIEW                       84
#define SQL_DIAG_DELETE_WHERE                      19
#define SQL_DIAG_DROP_ASSERTION                    24
#define SQL_DIAG_DROP_CHARACTER_SET                25
#define SQL_DIAG_DROP_COLLATION                    26
#define SQL_DIAG_DROP_DOMAIN                       27
#define SQL_DIAG_DROP_SCHEMA                       31
#define SQL_DIAG_DROP_TABLE                        32
#define SQL_DIAG_DROP_TRANSLATION                  33
#define SQL_DIAG_DROP_VIEW                         36
#define SQL_DIAG_DYNAMIC_DELETE_CURSOR             54
#define SQL_DIAG_DYNAMIC_UPDATE_CURSOR             55
#define SQL_DIAG_GRANT                             48
#define SQL_DIAG_INSERT                            50
```

```
#define SQL_DIAG_REVOKE                          59
#define SQL_DIAG_SELECT                          41
#define SQL_DIAG_SELECT_CURSOR                   85
#define SQL_DIAG_SET_CATALOG                     66
#define SQL_DIAG_SET_CONSTRAINT                  68
#define SQL_DIAG_SET_NAMES                       72
#define SQL_DIAG_SET_SCHEMA                      74
#define SQL_DIAG_SET_SESSION_AUTHORIZATION       76
#define SQL_DIAG_SET_TIME_ZONE                   71
#define SQL_DIAG_SET_TRANSACTION                 75
#define SQL_DIAG_UNKNOWN_STATEMENT                0
#define SQL_DIAG_UPDATE_WHERE                    82

/* SQL data type codes */
#define SQL_CHAR                                  1
#define SQL_NUMERIC                               2
#define SQL_DECIMAL                               3
#define SQL_INTEGER                               4
#define SQL_SMALLINT                              5
#define SQL_FLOAT                                 6
#define SQL_REAL                                  7
#define SQL_DOUBLE                                8
#define SQL_DATETIME                              9
#define SQL_INTERVAL                             10
#define SQL_VARCHAR                              12
#define SQL_BIT                                  14
#define SQL_BIT_VARYING                          15

/*      Concise codes for datetime and interval data types          */
#define SQL_TYPE_DATE                            91
#define SQL_TYPE_TIME                            92
#define SQL_TYPE_TIME_WITH_TIMEZONE              94
#define SQL_TYPE_TIMESTAMP                       93
#define SQL_TYPE_TIMESTAMP_WITH_TIMEZONE         95
#define SQL_INTERVAL_DAY                        103
#define SQL_INTERVAL_DAY_TO_HOUR                108
#define SQL_INTERVAL_DAY_TO_MINUTE              109
#define SQL_INTERVAL_DAY_TO_SECOND              110
#define SQL_INTERVAL_HOUR                       104
#define SQL_INTERVAL_HOUR_TO_MINUTE             111
#define SQL_INTERVAL_HOUR_TO_SECOND             112
#define SQL_INTERVAL_MINUTE                     105
#define SQL_INTERVAL_MINUTE_TO_SECOND           113
#define SQL_INTERVAL_MONTH                      102
#define SQL_INTERVAL_SECOND                     106
#define SQL_INTERVAL_YEAR                       101
#define SQL_INTERVAL_YEAR_TO_MONTH              107


/*      GetTypeInfo request for all data types  */
#define SQL_ALL_TYPES                             0

/* BindCol() and BindParam() default conversion code */
#define SQL_DEFAULT                              99

/*      GetData code indicating that the application parameter */
/*      descriptor specifies the data type */
#define SQL_ARD_TYPE                            -99

/* date/time type subcodes */
#define SQL_CODE_DATE                             1
#define SQL_CODE_TIME                             2
```

```
#define SQL_CODE_TIMESTAMP                      3
#define SQL_CODE_TIME_ZONE                      4
#define SQL_CODE_TIMESTAMP_ZONE                 5

/* interval qualifier codes */
#define SQL_DAY                                 3
#define SQL_DAY_TO_HOUR                         8
#define SQL_DAY_TO_MINUTE                       9
#define SQL_DAY_TO_SECOND                      10
#define SQL_HOUR                                4
#define SQL_HOUR_TO_MINUTE                     11
#define SQL_HOUR_TO_SECOND                     12
#define SQL_MINUTE                              5
#define SQL_MINUTE_TO_SECOND                   13
#define SQL_MONTH                               2
#define SQL_SECOND                              6
#define SQL_YEAR                                1
#define SQL_YEAR_TO_MONTH                       7

/* CLI option values */
#define SQL_FALSE                               0
#define SQL_TRUE                                1
#define SQL_NONSCROLLABLE                       0
#define SQL_SCROLLABLE                          1

/* Parameter mode values */
#define SQL_PARAM_MODE_IN                       1
#define SQL_PARAM_MODE_OUT                      4
#define SQL_PARAM_MODE_INOUT                    2
#define SQL_PARAM_MODE_NONE                     0

/* values of NULLABLE field in descriptor */
#define SQL_NO_NULLS                            0
#define SQL_NULLABLE                            1

/*      Values returned by GetTypeInfo for the SEARCHABLE column        */
#define SQL_PRED_NONE                           0
#define SQL_PRED_CHAR                           1
#define SQL_PRED_BASIC                          2

/* values of UNNAMED field in descriptor */
#define SQL_NAMED                               0
#define SQL_UNNAMED                             1

/* values of ALLOC_TYPE field in descriptor */
#define SQL_DESC_ALLOC_AUTO                     1
#define SQL_DESC_ALLOC_USER                     2

/* EndTran() options */
#define SQL_COMMIT                              0
#define SQL_ROLLBACK                            1

/* FreeStmt() options */
#define SQL_CLOSE                               0
#define SQL_DROP                                1
#define SQL_UNBIND                              2
#define SQL_RESET_PARAMS                        3

/* null handles returned by AllocHandle() */
#define SQL_NULL_HENV                           0
#define SQL_NULL_HDBC                           0
#define SQL_NULL_HSTMT                          0
```

```
#define SQL_NULL_HDESC                                  0

/*      GetFunctions values to identify CLI routines    */
#define SQL_API_SQLALLOCCONNECT                         1
#define SQL_API_SQLALLOCENV                             2
#define SQL_API_SQLALLOCHANDLE                       1001
#define SQL_API_SQLALLOCSTMT                            3
#define SQL_API_SQLBINDCOL                              4
#define SQL_API_SQLBINDPARAM                         1002
#define SQL_API_SQLCANCEL                               5
#define SQL_API_SQLCLOSECURSOR                       1003
#define SQL_API_SQLCOLATTRIBUTE                         6
#define SQL_API_SQLCONNECT                              7
#define SQL_API_SQLCOPYDESC                          1004
#define SQL_API_SQLDATASOURCES                         57
#define SQL_API_SQLDESCRIBECOL                          8
#define SQL_API_SQLDISCONNECT                           9
#define SQL_API_SQLENDTRAN                           1005
#define SQL_API_SQLERROR                               10
#define SQL_API_SQLEXECDIRECT                          11
#define SQL_API_SQLEXECUTE                             12
#define SQL_API_SQLFETCH                               13
#define SQL_API_SQLFETCHSCROLL                       1021
#define SQL_API_SQLFREECONNECT                         14
#define SQL_API_SQLFREEENV                             15
#define SQL_API_SQLFREEHANDLE                        1006
#define SQL_API_SQLFREESTMT                            16
#define SQL_API_SQLGETCONNECTATTR                    1007
#define SQL_API_SQLGETCURSORNAME                       17
#define SQL_API_SQLGETDATA                             43
#define SQL_API_SQLGETDESCFIELD                      1008
#define SQL_API_SQLGETDESCREC                        1009
#define SQL_API_SQLGETDIAGFIELD                      1010
#define SQL_API_SQLGETDIAGREC                        1011
#define SQL_API_SQLGETENVATTR                        1012
#define SQL_API_SQLGETFUNCTIONS                        44
#define SQL_API_SQLGETINFO                             45
#define SQL_API_SQLGETSTMTATTR                       1014
#define SQL_API_SQLGETTYPEINFO                         47
#define SQL_API_SQLNUMRESULTCOLS                       18
#define SQL_API_SQLPARAMDATA                           48
#define SQL_API_SQLPREPARE                             19
#define SQL_API_SQLPUTDATA                             49
#define SQL_API_SQLROWCOUNT                            20
#define SQL_API_SQLSETCONNECTATTR                    1016
#define SQL_API_SQLSETCURSORNAME                       21
#define SQL_API_SQLSETDESCFIELD                      1017
#define SQL_API_SQLSETDESCREC                        1018
#define SQL_API_SQLSETENVATTR                        1019
#define SQL_API_SQLSETSTMTATTR                       1020

/*      Information requested by GetInfo            */
#define SQL_MAXIMUM_DRIVER_CONNECTIONS                  0
#define SQL_MAXIMUM_CONCURRENT_ACTIVITIES               1
#define SQL_DATA_SOURCE_NAME                            2
#define SQL_FETCH_DIRECTION                             8
#define SQL_SERVER_NAME                                13
#define SQL_DBMS_NAME                                  17
#define SQL_DBMS_VERSION                               18
#define SQL_CURSOR_COMMIT_BEHAVIOR                     23
#define SQL_DATA_SOURCE_READ_ONLY                      25
#define SQL_DEFAULT_TRANSACTION_ISOLATION              26
#define SQL_IDENTIFIER_CASE                            28
```