

DB2  
for Linux, UNIX, and Windows



Version 9 Release 7



**Call Level Interface Guide and Reference, Volume 2**  
**Updated July, 2012**



DB2  
for Linux, UNIX, and Windows



Version 9 Release 7



**Call Level Interface Guide and Reference, Volume 2**  
**Updated July, 2012**

**Note**

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 551.

**Edition Notice**

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at [www.ibm.com/shop/publications/order](http://www.ibm.com/shop/publications/order)
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at [www.ibm.com/planetwide](http://www.ibm.com/planetwide)

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corporation 1993, 2012.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

**About this book . . . . . ix**

## **Chapter 1. CLI and ODBC function summary . . . . . 1**

Unicode functions (CLI) . . . . .	5
SQLAllocConnect function (CLI) - Allocate connection handle . . . . .	6
SQLAllocEnv function (CLI) - Allocate environment handle . . . . .	7
SQLAllocHandle function (CLI) - Allocate handle . . . . .	7
SQLAllocStmt function (CLI) - Allocate a statement handle . . . . .	10
SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator . . . . .	10
SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column . . . . .	16
SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter . . . . .	19
SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator . . . . .	22
SQLBrowseConnect function (CLI) - Get required attributes to connect to data source . . . . .	36
SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows . . . . .	41
SQLCancel function (CLI) - Cancel statement . . . . .	46
SQLCloseCursor function (CLI) - Close cursor and discard pending results . . . . .	48
SQLColAttribute function (CLI) - Return a column attribute . . . . .	50
SQLColAttributes function (CLI) - Get column attributes . . . . .	58
SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table . . . . .	59
SQLColumns function (CLI) - Get column information for a table. . . . .	63
SQLConnect function (CLI) - Connect to a data source . . . . .	69
SQLCopyDesc function (CLI) - Copy descriptor information between handles . . . . .	71
SQLCreateDb function (CLI) - Create a database . . . . .	74
SQLCreatePkg . . . . .	76
SQLDataSources function (CLI) - Get list of data sources . . . . .	77
SQLDescribeCol function (CLI) - Return a set of attributes for a column . . . . .	80
SQLDescribeParam function (CLI) - Return description of a parameter marker. . . . .	84
SQLDisconnect function (CLI) - Disconnect from a data source . . . . .	86
SQLDriverConnect function (CLI) - (Expanded) Connect to a data source . . . . .	88
SQLDropDb function (CLI) - Drop a database . . . . .	93
SQLEndTran function (CLI) - End transactions of a connection or an environment . . . . .	94
SQLError function (CLI) - Retrieve error information	98

SQLExecDirect function (CLI) - Execute a statement directly . . . . .	98
SQLExecute function (CLI) - Execute a statement	103
SQLExtendedBind function (CLI) - Bind an array of columns . . . . .	106
SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows) . . . . .	109
SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes . . . . .	110
SQLExtendedProcedures function (CLI) - Get list of procedure names . . . . .	114
SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure . . . . .	119
SQLFetch function (CLI) - Fetch next row . . . . .	125
SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns . . . . .	132
Cursor positioning rules for SQLFetchScroll() (CLI) . . . . .	138
SQLForeignKeys function (CLI) - Get the list of foreign key columns . . . . .	141
SQLFreeConnect function (CLI) - Free connection handle. . . . .	145
SQLFreeEnv function (CLI) - Free environment handle. . . . .	146
SQLFreeHandle function (CLI) - Free handle resources . . . . .	146
SQLFreeStmt function (CLI) - Free (or reset) a statement handle . . . . .	149
SQLGetConnectAttr function (CLI) - Get current attribute setting . . . . .	151
SQLGetConnectOption function (CLI) - Return current setting of a connect option . . . . .	154
SQLGetCursorName function (CLI) - Get cursor name . . . . .	154
SQLGetData function (CLI) - Get data from a column . . . . .	156
SQLGetDescField function (CLI) - Get single field settings of descriptor record . . . . .	162
SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record . . . . .	166
SQLGetDiagField function (CLI) - Get a field of diagnostic data . . . . .	170
SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record . . . . .	174
SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value . . . . .	177
SQLGetFunctions function (CLI) - Get functions	178
SQLGetInfo function (CLI) - Get general information . . . . .	180
SQLGetLength function (CLI) - Retrieve length of a string value . . . . .	210
SQLGetPosition function (CLI) - Return starting position of string . . . . .	212
SQLGetSQLCA function (CLI) - Get SQLCA data structure . . . . .	216

SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute . . . . .	216
SQLGetStmtOption function (CLI) - Return current setting of a statement option . . . . .	219
SQLGetSubString function (CLI) - Retrieve portion of a string value . . . . .	219
SQLGetTypeInfo function (CLI) - Get data type information . . . . .	222
SQLMoreResults function (CLI) - Determine if there are more result sets . . . . .	227
SQLNativeSql function (CLI) - Get native SQL text	229
SQLNumParams function (CLI) - Get number of parameters in a SQL statement . . . . .	230
SQLNextResult function (CLI) - Associate next result set with another statement handle . . . . .	232
SQLNumResultCols function (CLI) - Get number of result columns . . . . .	234
SQLParamData function (CLI) - Get next parameter for which a data value is needed . . . . .	236
SQLParamOptions function (CLI) - Specify an input array for a parameter. . . . .	238
SQLPrepare function (CLI) - Prepare a statement	239
SQLPrimaryKeys function (CLI) - Get primary key columns of a table. . . . .	243
SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure . . . . .	246
SQLProcedures function (CLI) - Get list of procedure names . . . . .	252
SQLPutData function (CLI) - Passing data value for a parameter . . . . .	256
SQLReloadConfig function (CLI) - Reload a configuration property from the client configuration file . . . . .	259
SQLRowCount function (CLI) - Get row count . . . . .	262
SQLSetColAttributes function (CLI) - Set column attributes . . . . .	264
SQLSetConnectAttr function (CLI) - Set connection attributes . . . . .	264
SQLSetConnection function (CLI) - Set connection handle. . . . .	267
SQLSetConnectOption function (CLI) - Set connection option . . . . .	269
SQLSetCursorName function (CLI) - Set cursor name . . . . .	269
SQLSetDescField function (CLI) - Set a single field of a descriptor record. . . . .	272
SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data . . . . .	276
SQLSetEnvAttr function (CLI) - Set environment attribute . . . . .	280
SQLSetParam function (CLI) - Bind a parameter marker to a buffer or LOB locator . . . . .	281
SQLSetPos function (CLI) - Set the cursor position in a rowset . . . . .	282
SQLSetStmtAttr function (CLI) - Set options related to a statement . . . . .	289
SQLSetStmtOption function (CLI) - Set statement option. . . . .	293
SQLSpecialColumns function (CLI) - Get special (row identifier) columns. . . . .	294

SQLStatistics function (CLI) - Get index and statistics information for a base table . . . . .	298
SQLTablePrivileges function (CLI) - Get privileges associated with a table . . . . .	303
SQLTables function (CLI) - Get table information	307
SQLTransact function (CLI) - Transaction management . . . . .	312

## Chapter 2. Return codes and SQLSTATES for CLI. . . . . 313

CLI function return codes . . . . .	313
SQLSTATES for CLI . . . . .	314
Return codes for compound SQL (CLI) in CLI applications . . . . .	315

## Chapter 3. CLI/ODBC configuration keywords listing by category . . . . 317

db2cli.ini initialization file . . . . .	322
AllowGetDataLOBReaccess CLI/ODBC configuration keyword . . . . .	325
AllowInterleavedGetData CLI/ODBC configuration keyword . . . . .	325
AltHostName CLI/ODBC configuration keyword	326
AltPort CLI/ODBC configuration keyword . . . . .	326
AppUsesLOBLocator CLI/ODBC configuration keyword . . . . .	327
AppendAPIName CLI/ODBC configuration keyword . . . . .	327
AppendForFetchOnly CLI/ODBC configuration keyword . . . . .	328
AppendRowColToErrorMessage CLI/ODBC configuration keyword . . . . .	328
ArrayInputChain CLI/ODBC configuration keyword . . . . .	329
AsyncEnable CLI/ODBC configuration keyword	330
Attach CLI/ODBC configuration keyword. . . . .	330
Authentication CLI/ODBC configuration keyword	331
AutoCommit CLI/ODBC configuration keyword	332
BIDI CLI/ODBC configuration keyword . . . . .	333
BitData CLI/ODBC configuration keyword . . . . .	333
BlockForNRows CLI/ODBC configuration keyword . . . . .	333
BlockLobs CLI/ODBC configuration keyword . . . . .	334
CLIPkg CLI/ODBC configuration keyword . . . . .	335
CheckForFork CLI/ODBC configuration keyword	335
ClientAcctStr CLI/ODBC configuration keyword	336
ClientApplName CLI/ODBC configuration keyword . . . . .	336
ClientBuffersUnboundLOBS CLI/ODBC configuration keyword . . . . .	337
ClientEncAlg CLI/ODBC configuration keyword	337
ClientUserID CLI/ODBC configuration keyword	338
ClientWrkStnName CLI/ODBC configuration keyword . . . . .	339
ColumnwiseMRI CLI/ODBC configuration keyword . . . . .	339
CommitOnEOF CLI/ODBC configuration keyword	340
ConcurrentAccessResolution CLI/ODBC configuration keyword . . . . .	340
ConnectNode CLI/ODBC configuration keyword	341

ConnectTimeout CLI/ODBC configuration keyword . . . . .	342	IgnoreWarnList CLI/ODBC configuration keyword	364
ConnectType CLI/ODBC configuration keyword	343	IgnoreWarnings CLI/ODBC configuration keyword	364
CurrentFunctionPath CLI/ODBC configuration keyword . . . . .	343	Instance CLI/ODBC configuration keyword . . .	364
CurrentImplicitXMLParseOption CLI/ODBC configuration keyword . . . . .	344	Interrupt CLI/ODBC configuration keyword . . .	365
CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword . . . . .	344	KRBPlugin CLI/ODBC configuration keyword . .	365
CURRENTOPTIMIZATIONPROFILE CLI/ODBC configuration keyword . . . . .	345	KeepDynamic CLI/ODBC configuration keyword	366
CurrentPackagePath CLI/ODBC configuration keyword . . . . .	345	LOBCacheSize CLI/ODBC configuration keyword	366
CurrentPackageSet CLI/ODBC configuration keyword . . . . .	346	LOBFileThreshold CLI/ODBC configuration keyword . . . . .	367
CurrentRefreshAge CLI/ODBC configuration keyword . . . . .	346	LOBMaxColumnSize CLI/ODBC configuration keyword . . . . .	367
CurrentSQLID CLI/ODBC configuration keyword	347	LoadXAInterceptor CLI/ODBC configuration keyword . . . . .	368
CurrentSchema CLI/ODBC configuration keyword	347	LockTimeout CLI/ODBC configuration keyword	368
CursorHold CLI/ODBC configuration keyword	347	LongDataCompat CLI/ODBC configuration keyword . . . . .	368
CursorTypes CLI/ODBC configuration keyword	348	MapBigintCDefault CLI/ODBC configuration keyword . . . . .	369
DB2Degree CLI/ODBC configuration keyword . .	348	MapCharToWChar CLI/ODBC configuration keyword . . . . .	369
DB2Explain CLI/ODBC configuration keyword	349	MapDateCDefault CLI/ODBC configuration keyword . . . . .	370
DB2NETNamedParam CLI/ODBC configuration keyword . . . . .	350	MapDateDescribe CLI/ODBC configuration keyword . . . . .	370
DB2Optimization CLI/ODBC configuration keyword . . . . .	350	MapDecimalFloatDescribe CLI/ODBC configuration keyword . . . . .	371
DBAlias CLI/ODBC configuration keyword . . .	351	MapGraphicDescribe CLI/ODBC configuration keyword . . . . .	372
DBName CLI/ODBC configuration keyword . . .	351	MapTimeCDefault CLI/ODBC configuration keyword . . . . .	372
DSN CLI/ODBC configuration keyword . . . .	352	MapTimeDescribe CLI/ODBC configuration keyword . . . . .	373
Database CLI/ODBC configuration keyword . . .	352	MapTimestampCDefault CLI/ODBC configuration keyword . . . . .	373
DateTimeStringFormat CLI/ODBC configuration keyword . . . . .	352	MapTimestampDescribe CLI/ODBC configuration keyword . . . . .	374
DecimalFloatRoundingMode CLI/ODBC configuration keyword . . . . .	353	MapXMLCDefault CLI/ODBC configuration keyword . . . . .	375
DeferredPrepare CLI/ODBC configuration keyword . . . . .	354	MapXMLDescribe CLI/ODBC configuration keyword . . . . .	375
DescribeCall CLI/ODBC configuration keyword	355	MaxLOBBlockSize CLI/ODBC configuration keyword . . . . .	376
DescribeInputOnPrepare CLI/ODBC configuration keyword . . . . .	355	Mode CLI/ODBC configuration keyword . . . .	376
DescribeOutputLevel CLI/ODBC configuration keyword . . . . .	356	NotifyLevel CLI/ODBC configuration keyword	376
DescribeParam CLI/ODBC configuration keyword	357	OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword . . . . .	377
DiagLevel CLI/ODBC configuration keyword . .	358	OleDbReturnCharAsWChar CLI/ODBC configuration keyword . . . . .	377
DiagPath CLI/ODBC configuration keyword . .	358	OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword . . . . .	378
DisableKeysetCursor CLI/ODBC configuration keyword . . . . .	358	OnlyUseBigPackages CLI/ODBC configuration keyword . . . . .	379
DisableMultiThread CLI/ODBC configuration keyword . . . . .	358	OptimizeForNRows CLI/ODBC configuration keyword . . . . .	379
DisableUnicode CLI/ODBC configuration keyword	359	PWD CLI/ODBC configuration keyword . . . .	379
EnableNamedParameterSupport CLI/ODBC configuration keyword . . . . .	359	PWDPlugin CLI/ODBC configuration keyword	380
FET_BUF_SIZE CLI/ODBC configuration keyword	360	Patch1 CLI/ODBC configuration keyword. . . .	380
FileDSN CLI/ODBC configuration keyword . . .	360	Patch2 CLI/ODBC configuration keyword. . . .	383
FloatPrecRadix CLI/ODBC configuration keyword	360	Port CLI/ODBC configuration keyword . . . .	386
GetDataLobNoTotal CLI/ODBC configuration keyword . . . . .	361	ProgramID CLI/ODBC configuration keyword . .	387
GranteeList CLI/ODBC configuration keyword . .	362	ProgramName CLI/ODBC configuration keyword	387
GrantorList CLI/ODBC configuration keyword . .	362		
Graphic CLI/ODBC configuration keyword . . .	363		
Hostname CLI/ODBC configuration keyword . .	363		

PromoteLONGVARtoLOB CLI/ODBC configuration keyword . . . . .	388
Protocol CLI/ODBC configuration keyword . . . . .	388
QueryTimeoutInterval CLI/ODBC configuration keyword . . . . .	389
ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword . . . . .	390
ReceiveTimeout CLI/ODBC configuration keyword . . . . .	390
Reopt CLI/ODBC configuration keyword . . . . .	390
ReportPublicPrivileges CLI/ODBC configuration keyword . . . . .	391
ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword . . . . .	391
RetCatalogAsCurrServer CLI/ODBC configuration keyword . . . . .	392
RetOleDbConnStr CLI/ODBC configuration keyword . . . . .	392
RetryOnError CLI/ODBC configuration keyword . . . . .	393
ReturnAliases CLI/ODBC configuration keyword . . . . .	394
ReturnSynonymSchema CLI/ODBC configuration keyword . . . . .	394
SQLOverrideFileName CLI/ODBC configuration keyword . . . . .	395
SaveFile CLI/ODBC configuration keyword . . . . .	396
SchemaList CLI/ODBC configuration keyword . . . . .	396
security CLI/ODBC configuration keyword . . . . .	397
ServerMsgMask CLI/ODBC configuration keyword . . . . .	397
ServiceName CLI/ODBC configuration keyword . . . . .	398
SkipTrace CLI/ODBC configuration keyword . . . . .	398
SQLCODEMAP CLI/ODBC configuration keyword . . . . .	399
SSLClientLabel CLI/ODBC configuration keyword . . . . .	399
SSLClientKeystash CLI/ODBC configuration keyword . . . . .	400
SSLClientKeystoredb CLI/ODBC configuration keyword . . . . .	400
SSLClientKeystoreDBPassword CLI/ODBC configuration keyword . . . . .	401
StaticCapFile CLI/ODBC configuration keyword . . . . .	401
StaticLogFile CLI/ODBC configuration keyword . . . . .	402
StaticMode CLI/ODBC configuration keyword . . . . .	402
StaticPackage CLI/ODBC configuration keyword . . . . .	402
StmtConcentrator CLI/ODBC configuration keyword . . . . .	403
StreamGetData CLI/ODBC configuration keyword . . . . .	403
StreamPutData CLI/ODBC configuration keyword . . . . .	404
SysSchema CLI/ODBC Configuration Keyword . . . . .	404
TableType CLI/ODBC configuration keyword . . . . .	405
TargetPrincipal CLI/ODBC configuration keyword . . . . .	406
TempDir CLI/ODBC configuration keyword . . . . .	406
TimestampTruncErrToWarning CLI/ODBC configuration keyword . . . . .	407
Trace CLI/ODBC configuration keyword . . . . .	407
TraceAPIList CLI/ODBC configuration keyword . . . . .	408
TraceAPIList! CLI/ODBC configuration keyword . . . . .	410
TraceComm CLI/ODBC configuration keyword . . . . .	412
TraceErrImmediate CLI/ODBC configuration keyword . . . . .	413
TraceFileName CLI/ODBC configuration keyword . . . . .	413
TraceFlush CLI/ODBC configuration keyword . . . . .	414
TraceFlushOnError CLI/ODBC configuration keyword . . . . .	415

TraceLocks CLI/ODBC configuration keyword . . . . .	415
TracePIDList CLI/ODBC configuration keyword . . . . .	416
TracePIDTHID CLI/ODBC configuration keyword . . . . .	416
TracePathName CLI/ODBC configuration keyword . . . . .	417
TraceRefreshInterval CLI/ODBC configuration keyword . . . . .	418
TraceStmtOnly CLI/ODBC configuration keyword . . . . .	418
TraceTime CLI/ODBC configuration keyword . . . . .	419
TraceTimestamp CLI/ODBC configuration keyword . . . . .	419
Trusted_Connection CLI/ODBC configuration keyword . . . . .	420
TxnIsolation CLI/ODBC configuration keyword . . . . .	421
UID CLI/ODBC configuration keyword . . . . .	422
Underscore CLI/ODBC configuration keyword . . . . .	422
UseOldStpCall CLI/ODBC configuration keyword . . . . .	423
UseServerMsgSP CLI/ODBC configuration keyword . . . . .	423
ServerMsgTextSP CLI/ODBC configuration keyword . . . . .	424
WarningList CLI/ODBC configuration keyword . . . . .	424
XMLDeclaration CLI/ODBC configuration keyword . . . . .	425

<b>Chapter 4. Environment, connection, and statement attributes in CLI applications . . . . .</b>	<b>427</b>
Environment attributes (CLI) list . . . . .	429
Connection attributes (CLI) list . . . . .	436
Statement attributes (CLI) list . . . . .	466

<b>Chapter 5. Descriptor values . . . . .</b>	<b>489</b>
Descriptor FieldIdentifier argument values (CLI) . . . . .	489
Descriptor header and record field initialization values (CLI) . . . . .	500

<b>Chapter 6. Header and record fields for the DiagIdentifier argument (CLI) . . . . .</b>	<b>505</b>
--	------------

<b>Chapter 7. CLI data type attributes . . . . .</b>	<b>511</b>
SQL symbolic and default data types for CLI applications . . . . .	511
C data types for CLI applications. . . . .	512
Data conversions supported in CLI . . . . .	517
SQL to C data conversion in CLI . . . . .	520
C to SQL data conversion in CLI . . . . .	527
Data type attributes . . . . .	532
Data type precision (CLI) table . . . . .	532
Data type scale (CLI) table . . . . .	534
Data type length (CLI) table . . . . .	535
Data type display (CLI) table . . . . .	536

<b>Appendix A. Overview of the DB2 technical information . . . . .</b>	<b>539</b>
DB2 technical library in hardcopy or PDF format . . . . .	539
Ordering printed DB2 books . . . . .	542
Displaying SQL state help from the command line processor . . . . .	543



Accessing different versions of the DB2 Information Center . . . . .	543
Displaying topics in your preferred language in the DB2 Information Center . . . . .	544
Updating the DB2 Information Center installed on your computer or intranet server . . . . .	544
Manually updating the DB2 Information Center installed on your computer or intranet server . . . . .	546

DB2 tutorials . . . . .	547
DB2 troubleshooting information . . . . .	548
Terms and Conditions . . . . .	548

**Appendix B. Notices . . . . . 551**

**Index . . . . . 555**



---

## About this book

The Call Level Interface (CLI) Guide and Reference is in two volumes:

- Volume 1 describes how to use CLI to create database applications for DB2<sup>®</sup> Database for Linux, UNIX, and Windows.
- Volume 2 is a reference that describes CLI functions, keywords and configuration.

## About this book

## Chapter 1. CLI and ODBC function summary

**Depr** in the ODBC column indicates that the function has been deprecated for ODBC.

The SQL/CLI column can have the following values:

**95** The function is defined in the SQL/CLI 9075-3 specification.

**SQL3** The function is defined in the SQL/CLI part of the ISO SQL3 draft replacement for SQL/CLI 9075-3.

Table 1. CLI Function list by category

Task Function name	ODBC 3.0	SQL/ CLI	DB2 CLI first version supported	Purpose
Connecting to a data source				
SQLConnect()	Depr	95	V1.1	Obtains a connection handle.
SQLAllocEnv()	Depr	95	V1.1	Obtains an environment handle. One environment handle is used for one or more connections.
SQLAllocHandle()	Core	95	V5	Obtains a handle.
SQLBrowseConnect()	Level 1	95	V5	Gets required attributes to connect to a data source.
SQLConnect()	Core	95	V1.1	Connects to a specific driver by using a data source name, user ID, and password.
SQLDriverConnect()	Core	SQL3	V2.1 <sup>1</sup>	Connects to a specific driver by using a connection string or optionally requests that the Driver Manager and driver display connection dialogs for the user. <b>Note:</b> This function is also affected by the additional IBM® keywords supported in the ODBC.INI file.
SQLDrivers()	Core	No	None	CLI does not support this function because this function is implemented by a Driver Manager.
SQLSetConnectAttr()	Core	95	V5	Sets connection attributes.
SQLSetConnectOption()	Depr	95	V2.1	Sets connection attributes.
SQLSetConnection()	No	SQL3	V2.1	Sets the current active connection. You have to use this function only when using embedded SQL within a CLI application with multiple concurrent connections.
Obtaining information about a driver and data source				
SQLDataSources()	Lvl 2	95	V1.1	Returns the list of available data sources.
SQLGetInfo()	Core	95	V1.1	Returns information about a specific driver and data source.
SQLGetFunctions()	Core	95	V1.1	Returns a list of supported driver functions.

## CLI and ODBC function summary

Table 1. CLI Function list by category (continued)

Task Function name	ODBC 3.0	SQL/ CLI	DB2 CLI first version supported	Purpose
SQLGetTypeInfo()	Core	95	V1.1	Returns information about supported data types.
Setting and retrieving driver options				
SQLCreatePkg()	No	No	V9.5	Binds packages to the database.
SQLSetEnvAttr()	Core	95	V2.1	Sets an environment option.
SQLGetEnvAttr()	Core	95	V2.1	Returns the value of an environment option.
SQLGetConnectAttr()	Lvl 1	95	V5	Returns the value of a connection option.
SQLGetConnectOption()	Depr	95	V2.1 <sup>1</sup>	Returns the value of a connection option.
SQLSetStmtAttr()	Core	95	V5	Sets a statement attribute.
SQLSetStmtOption()	Depr	95	V2.1 <sup>1</sup>	Sets a statement option.
SQLGetStmtAttr()	Core	95	V5	Returns the value of a statement attribute.
SQLGetStmtOption()	Depr	95	V2.1 <sup>1</sup>	Returns the value of a statement option.
SQLReloadConfig()	No	No	V9.7	Reloads a configuration property from the client configuration file <code>db2dsdriver.cfg</code>
Preparing SQL requests				
SQLAllocStmnt()	Depr	95	V1.1	Allocates a statement handle.
SQLPrepare()	Core	95	V1.1	Prepares an SQL statement for later execution.
SQLExtendedPrepare()	No	No	V6	Prepares an array of statement attributes for an SQL statement for later execution.
SQLExtendedBind()	No	No	V6	Bind an array of columns instead of using repeated calls to <code>SQLBindCol()</code> and <code>SQLBindParameter()</code>
SQLBindParameter()	Lvl 1	95 <sup>2</sup>	V2.1	Assigns storage for a parameter in an SQL statement (ODBC 2.0).
SQLSetParam()	Depr	No	V1.1	Assigns storage for a parameter in an SQL statement (ODBC 1.0). <b>Note:</b> In ODBC 2.0, this function has been replaced by <code>SQLBindParameter()</code> .
SQLParamOptions()	Depr	No	V2.1	Specifies the use of multiple values for parameters.
SQLGetCursorName()	Core	95	V1.1	Returns the cursor name associated with a statement handle.
SQLSetCursorName()	Core	95	V1.1	Specifies a cursor name.
Submitting requests				
SQLDescribeParam()	Level 2	SQL3	V5	Returns the description of a parameter marker.
SQLExecute()	Core	95	V1.1	Executes a prepared statement.
SQLExecDirect()	Core	95	V1.1	Executes a statement.
SQLNativeSql()	Lvl 2	95	V2.1 <sup>1</sup>	Returns the text of an SQL statement as translated by the driver.

*Table 1. CLI Function list by category (continued)*

<b>Task Function name</b>	<b>ODBC 3.0</b>	<b>SQL/ CLI</b>	<b>DB2 CLI first version supported</b>	<b>Purpose</b>
SQLNumParams()	Lvl 2	95	V2.1 <sup>1</sup>	Returns the number of parameters in a statement.
SQLParamData()	Lvl 1	95	V2.1 <sup>1</sup>	Used in conjunction with SQLPutData() to supply parameter data at execution time. This is useful for long data values.
SQLPutData()	Core	95	V2.1 <sup>1</sup>	Sends part or all of a data value for a parameter. This is useful for long data values.
Retrieving results and information about results				
SQLRowCount()	Core	95	V1.1	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols()	Core	95	V1.1	Returns the number of columns in the result set.
SQLDescribeCol()	Core	95	V1.1	Describes a column in the result set.
SQLColAttribute()	Core	Yes	V5	Describes attributes of a column in the result set.
SQLColAttributes()	Depr	Yes	V1.1	Describes attributes of a column in the result set.
SQLColumnPrivileges()	Level 2	95	V2.1	Gets privileges associated with the columns of a table.
SQLSetColAttributes()	No	No	V2.1	Sets attributes of a column in the result set.
SQLBindCol()	Core	95	V1.1	Assigns storage for a result column and specifies the data type.
SQLFetch()	Core	95	V1.1	Returns a result row.
SQLFetchScroll()	Core	95	V5	Returns a rowset from a result row.
SQLExtendedFetch()	Depr	95	V2.1	Returns multiple result rows.
SQLGetData()	Core	95	V1.1	Returns part or all of one column of one row of a result set. This is useful for long data values.
SQLMoreResults()	Lvl 1	SQL3	V2.1 <sup>a</sup>	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLNextResult()	No	Yes	V7.1	Provides nonsequential access to multiple result sets returned from a stored procedure.
SQLError()	Depr	95	V1.1	Returns additional error or status information.
SQLGetDiagField()	Core	95	V5	Gets a field of diagnostic data.
SQLGetDiagRec()	Core	95	V5	Gets multiple fields of diagnostic data.
SQLSetPos()	Level 1	SQL3	V5	Sets the cursor position in a rowset.
SQLGetSQLCA()	No	No	V2.1	Returns the SQLCA associated with a statement handle.
SQLBulkOperations()	Level 1	No	V6	Performs bulk insertions, updates, deletions, and fetches by bookmark.

## CLI and ODBC function summary

Table 1. CLI Function list by category (continued)

Task Function name	ODBC 3.0	SQL/ CLI	DB2 CLI first version supported	Purpose
Descriptors				
SQLCopyDesc()	Core	95	V5	Copies descriptor information between handles.
SQLGetDescField()	Core	95	V5	Gets single field settings of a descriptor record.
SQLGetDescRec()	Core	95	V5	Gets multiple field settings of a descriptor record.
SQLSetDescField()	Core	95	V5	Sets a single field of a descriptor record.
SQLSetDescRec()	Core	95	V5	Sets multiple field settings of a descriptor record.
Large object support				
SQLBindFileToCol()	No	No	V2.1	Associates a LOB file reference with a LOB column.
SQLBindFileToParam()	No	No	V2.1	Associates a LOB file reference with a parameter marker.
SQLGetLength()	No	SQL3	V2.1	Gets the length of a string referenced by a LOB locator.
SQLGetPosition()	No	SQL3	V2.1	Gets the position of a string within a source string referenced by a LOB locator.
SQLGetSubString()	No	SQL3	V2.1	Creates a new LOB locator that references a substring within a source string. The source string is also represented by a LOB locator.
Obtaining information about the data source's system tables (catalog functions)				
SQLColumns()	Lvl 1	SQL3	V2.1 <sup>1</sup>	Returns the list of column names in specified tables.
SQLExtendedProcedures()	No	No	V9.7	Returns the list of procedure names stored in a specific data source with additional information.
SQLExtendedProceduresColumns()	No	No	V9.7	Returns the list of input and output parameters for the specified procedures with additional information.
SQLForeignKeys()	Lvl 2	SQL3	V2.1	Returns the list of column names that comprise foreign keys, if they exist for a specified table.
SQLPrimaryKeys()	Lvl 1	SQL3	V2.1	Returns the list of column names that comprise the primary key for a table.
SQLProcedureColumns()	Lvl 2	No	V2.1	Returns the list of input and output parameters for the specified procedures.
SQLProcedures()	Lvl 2	No	V2.1	Returns the list of procedure names stored in a specific data source.
SQLSpecialColumns()	Core	SQL3	V2.1 <sup>1</sup>	Returns information about the optimal set of columns that uniquely identifies a row in a specified table.



*Table 1. CLI Function list by category (continued)*

Task Function name	ODBC 3.0	SQL/ CLI	DB2 CLI first version supported	Purpose
SQLStatistics()	Core	SQL3	V2.1 <sup>1</sup>	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges()	Lvl 2	SQL3	V2.1	Returns a list of tables and the privileges associated with each table.
SQLTables()	Core	SQL3	V2.1 <sup>1</sup>	Returns the list of table names stored in a specific data source.
<b>Terminating a statement</b>				
SQLFreeHandle()	Core	95	V1.1	Frees handle resources.
SQLFreeStmt()	Core	95	V1.1	Ends statement processing and closes the associated cursor, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCancel()	Core	95	V1.1	Cancels an SQL statement.
SQLTransact()	Depr	No	V1.1	Commits or rolls back a transaction.
SQLCloseCursor()	Core	95	V5	Commits or rolls back a transaction.
<b>Terminating a connection</b>				
SQLDisconnect()	Core	95	V1.1	Closes the connection.
SQLEndTran()	Core	95	V5	Ends the transaction of a connection.
SQLFreeConnect()	Depr	95	V1.1	Releases the connection handle.
SQLFreeEnv()	Depr	95	V1.1	Releases the environment handle.
<b>Creating and dropping a database</b>				
SQLCreateDb()	No	No	V9.7	Creates a database based on the specified database name, code-set, and mode.
SQLDropDb()	No	No	V9.7	Drops the specified database.

**Note:**

<sup>1</sup> Runtime support for this function was also available in the DB2 Client Application Enabler for DOS Version 1.2 product.

<sup>2</sup> SQLBindParam() has been replaced by SQLBindParameter().

The following limitations apply to ODBC functions:

- SQLSetScrollOptions() is supported for runtime use only, because it has been superseded by the SQL\_CURSOR\_TYPE, SQL\_CONCURRENCY, SQL\_KEYSET\_SIZE, and SQL\_ROWSET\_SIZE statement options.
- SQLDrivers() is implemented by the ODBC driver manager.

## Unicode functions (CLI)

CLI Unicode functions accept Unicode string arguments in place of ANSI string arguments. The Unicode string arguments must be in UCS-2 encoding (native-endian format). ODBC API functions have suffixes to indicate the format of their string arguments: those that accept Unicode end in W, and those that accept ANSI have no suffix (ODBC adds equivalent functions with names that end in A, but these are not offered by CLI). The following is a list of functions available in CLI which have both ANSI and Unicode versions:

## Unicode functions (CLI)

SQLBrowseConnect	SQLGetDiagField
SQLColAttribute	SQLGetDiagRec
SQLColAttributes	SQLGetInfo
SQLColumnPrivileges	SQLGetPosition
SQLColumns	SQLGetStmtAttr
SQLConnect	SQLNativeSQL
SQLCreateDb	SQLPrepare
SQLDataSources	SQLPrimaryKeys
SQLDescribeCol	SQLProcedureColumns
SQLDriverConnect	SQLProcedures
SQLDropDb	SQLReloadConfig
SQLError	SQLSetConnectAttr
SQLExecDirect	SQLSetConnectOption
SQLExtendedPrepare	SQLSetCursorName
SQLExtendedProcedures	SQLSetDescField
SQLExtendedProcedureColumns	SQLSetStmtAttr
SQLForeignKeys	SQLSpecialColumns
SQLGetConnectAttr	SQLStatistics
SQLGetConnectOption	SQLTablePrivileges
SQLGetCursorName	SQLTables
SQLGetDescField	
SQLGetDescRec	

Unicode functions that have arguments which are always the length of strings interpret these arguments as the number of SQLWCHAR elements needed to store the string. For functions that return length information for server data, the display size and precision are again described in terms of the number of SQLWCHAR elements used to store them. When the length (transfer size of the data) could refer to string or non-string data, it is interpreted as the number of bytes needed to store the data.

For example, `SQLGetInfoW()` will still take the length as the number of bytes, but `SQLExecDirectW()` will use the number of SQLWCHAR elements. Consider a single character from the UTF-16 extended character set (UTF-16 is an extended character set of UCS-2; Microsoft Windows 2000 and Microsoft Windows XP use UTF-16). Microsoft Windows 2000 will use two `SQL_C_WCHAR` elements, which is equivalent to 4 bytes, to store this single character. The character therefore has a display size of 1, a string length of 2 (when using `SQL_C_WCHAR`), and a byte count of 4. CLI will return data from result sets in either Unicode or ANSI, depending on the application's binding. If an application binds to `SQL_C_CHAR`, the driver will convert `SQL_WCHAR` data to `SQL_CHAR`. An ODBC driver manager, if used, maps `SQL_C_WCHAR` to `SQL_C_CHAR` for ANSI drivers but does no mapping for Unicode drivers.

### ANSI to Unicode function mappings

The syntax for a CLI Unicode function is the same as the syntax for its corresponding ANSI function, except that `SQLCHAR` parameters are defined as `SQLWCHAR`. Character buffers defined as `SQLPOINTER` in the ANSI syntax can be defined as either `SQLCHAR` or `SQLWCHAR` in the Unicode function. Refer to the ANSI version of the CLI Unicode functions for ANSI syntax details.

---

## SQLAllocConnect function (CLI) - Allocate connection handle

### Deprecated

**Note:**

## SQLAllocConnect function (CLI) - Allocate connection handle

In ODBC 3.0, `SQLAllocConnect()` has been deprecated and replaced with `SQLAllocHandle()`.

Although this version of CLI continues to support `SQLAllocConnect()`, it is recommended that you use `SQLAllocHandle()` in your CLI programs so that they conform to the latest standards.

### Migrating to the new function

The statement:

```
SQLAllocConnect(henv, &hdbc);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
```

---

## SQLAllocEnv function (CLI) - Allocate environment handle

### Deprecated

**Note:**

In ODBC 3.0, `SQLAllocEnv()` has been deprecated and replaced with `SQLAllocHandle()`.

Although this version of CLI continues to support `SQLAllocEnv()`, use `SQLAllocHandle()` in your CLI programs so that they conform to the latest standards.

### Migrating to the new function

The statement:

```
SQLAllocEnv(&henv);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```

---

## SQLAllocHandle function (CLI) - Allocate handle

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

`SQLAllocHandle()` is a generic function that allocates environment, connection, statement, or descriptor handles.

**Note:** This function replaces the deprecated ODBC 2.0 functions `SQLAllocConnect()`, `SQLAllocEnv()`, and `SQLAllocStmt()`.

### Syntax

```
SQLRETURN SQLAllocHandle (
    SQLSMALLINT HandleType, /* fHandleType */
    SQLHANDLE InputHandle, /* hInput */
    SQLHANDLE *OutputHandlePtr); /* *phOutput */
```

## SQLAllocHandle function (CLI) - Allocate handle

### Function Arguments

Table 2. SQLAllocHandle arguments

Data type	Argument	Use	Description
SQLSMALLINT	<i>HandleType</i>	input	The type of handle to be allocated by SQLAllocHandle(). Must be one of the following values: <ul style="list-style-type: none"><li>• SQL_HANDLE_ENV</li><li>• SQL_HANDLE_DBC</li><li>• SQL_HANDLE_STMT</li><li>• SQL_HANDLE_DESC</li></ul>
SQLHANDLE	<i>InputHandle</i>	input	Existing handle to use as a context for the new handle being allocated. If HandleType is SQL_HANDLE_ENV, this is SQL_NULL_HANDLE. If HandleType is SQL_HANDLE_DBC, this must be an environment handle, and if it is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle.
SQLHANDLE *	<i>OutputHandlePtr</i>	output	Pointer to a buffer in which to return the handle to the newly allocated data structure.

### Usage

SQLAllocHandle() is used to allocate environment, connection, statement, and descriptor handles. An application can allocate multiple environment, connection, statement, or descriptor handles at any time a valid *InputHandle* exists.

If the application calls SQLAllocHandle() with *\*OutputHandlePtr* set to an existing environment, connection, statement, or descriptor handle, CLI overwrites the handle, and new resources appropriate to the handle type are allocated. There are no changes made to the CLI resources associated with the original handle.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_INVALID\_HANDLE
- SQL\_ERROR

If SQLAllocHandle() returns SQL\_INVALID\_HANDLE, it will set *OutputHandlePtr* to SQL\_NULL\_HENV, SQL\_NULL\_HDBC, SQL\_NULL\_HSTMT, or SQL\_NULL\_HDESC, depending on the value of *HandleType*, unless the output argument is a null pointer. The application can then obtain additional information from the diagnostic data structure associated with the handle in the *InputHandle* argument.

### Diagnostics

Table 3. SQLAllocHandle SQLSTATES

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)

## SQLAllocHandle function (CLI) - Allocate handle

Table 3. SQLAllocHandle SQLSTATES (continued)

SQLSTATE	Description	Explanation
08003	Connection is closed.	The HandleType argument was SQL_HANDLE_STMT or SQL_HANDLE_DESC, but the connection handle specified by the InputHandle argument did not have an open connection. The connection process must be completed successfully (and the connection must be open) for CLI to allocate a statement or descriptor handle.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	The HandleType argument was SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC; and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY014	No more handles.	The limit for the number of handles that can be allocated for the type of handle indicated by the HandleType argument has been reached, or in some cases, insufficient system resources exist to properly initialize the new handle.
HY092	Option type out of range.	The HandleType argument was not one of: <ul style="list-style-type: none"><li>• SQL_HANDLE_ENV</li><li>• SQL_HANDLE_DBC</li><li>• SQL_HANDLE_STMT</li><li>• SQL_HANDLE_DESC</li></ul>

### Restrictions

None.

### Example

```
SQLHANDLE henv; /* environment handle */
SQLHANDLE hdbc; /* connection handle */
SQLHANDLE hstmt; /* statement handle */
SQLHANDLE hdesc; /* descriptor handle */

/* ... */

/* allocate an environment handle */
cliRC = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

/* ... */

/* allocate a database connection handle */
cliRC = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

/* ... */
/* connect to database using hdbc */
/* ... */

/* allocate one or more statement handles */
cliRC = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

## SQLAllocHandle function (CLI) - Allocate handle

```
/* ... */
/* allocate a descriptor handle */
cliRC = SQLAllocHandle(SQL_HANDLE_DESC, hstmt, &hdesc);
```

---

## SQLAllocStmt function (CLI) - Allocate a statement handle

### Deprecated

#### Note:

In ODBC 3.0, `SQLAllocStmt()` has been deprecated and replaced with `SQLAllocHandle()`.

Although this version of CLI continues to support `SQLAllocStmt()`, use `SQLAllocHandle()` in your CLI programs so that they conform to the latest standards.

#### Migrating to the new function

The statement:

```
SQLAllocStmt(hdbc, &hstmt);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

---

## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

`SQLBindCol()` is used to associate (bind) columns in a result set to either:

- Application variables or arrays of application variables (storage buffers), for all C data types. Data is transferred from the DBMS to the application when `SQLFetch()` or `SQLFetchScroll()` is called. Data conversion might occur as the data is transferred.
- A LOB locator, for LOB columns. A LOB locator, not the data itself, is transferred from the DBMS to the application when `SQLFetch()` is called.

Alternatively, LOB columns can be bound directly to a file using `SQLBindFileToCol()`.

`SQLBindCol()` is called once for each column in the result set that the application needs to retrieve.

In general, `SQLPrepare()`, `SQLExecDirect()` or one of the schema functions is called before this function, and `SQLFetch()`, `SQLFetchScroll()`, `SQLBulkOperations()`, or `SQLSetPos()` is called after. Column attributes might also be needed before calling `SQLBindCol()`, and can be obtained using `SQLDescribeCol()` or `SQLColAttribute()`.

### Syntax

```
SQLRETURN SQLBindCol (
    SQLHSTMT          StatementHandle,    /* hstmt */
    SQLUSMALLINT      ColumnNumber,       /* icol */
    SQLSMALLINT       TargetType,         /* fCType */
    ...)
```

## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

```

SQLPOINTER TargetValuePtr, /* rgbValue */
SQLLEN BufferLength, /* dbValueMax */
SQLLEN *StrLen_or_IndPtr); /* *pcbValue */
    
```

### Function arguments

Table 4. SQLBindCol arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLUSMALLINT	<i>ColumnNumber</i>	input	<p>Number identifying the column. Columns are numbered sequentially, from left to right.</p> <ul style="list-style-type: none"> <li>• Column numbers start at 1 if bookmarks are not used (SQL_ATTR_USE_BOOKMARKS statement attribute set to SQL_UB_OFF).</li> <li>• Column numbers start at 0 if bookmarks are used (the statement attribute is set to SQL_UB_ON). Column 0 is the bookmark column.</li> </ul>
SQLSMALLINT	<i>TargetType</i>	input	<p>The C data type for column number <i>ColumnNumber</i> in the result set. When the application retrieves data from the data source, it will convert the data to this C type. When using SQLBulkOperations() or SQLSetPos(), the driver will convert data from this C data type when sending information to the data source. The following types are supported:</p> <ul style="list-style-type: none"> <li>• SQL_C_BINARY</li> <li>• SQL_C_BIT</li> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CHAR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCHAR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> <li>• SQL_C_DECIMAL_IBM</li> <li>• SQL_C_DOUBLE</li> <li>• SQL_C_FLOAT</li> <li>• SQL_C_LONG</li> <li>• SQL_C_NUMERIC<sup>a</sup></li> <li>• SQL_C_SBIGINT</li> <li>• SQL_C_SHORT</li> <li>• SQL_C_TYPE_DATE</li> <li>• SQL_C_TYPE_TIME</li> <li>• SQL_C_TYPE_TIMESTAMP</li> <li>• SQL_C_TYPE_TIMESTAMP_EXT</li> <li>• SQL_C_TINYINT</li> <li>• SQL_C_UBIGINT</li> <li>• SQL_C_UTINYINT</li> <li>• SQL_C_WCHAR</li> </ul> <p>Specifying SQL_C_DEFAULT causes data to be transferred to its default C data type.</p>

## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

Table 4. SQLBindCol arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	<i>TargetValuePtr</i>	input/output (deferred)	<p>Pointer to buffer or an array of buffers with either column-wise or row-wise binding, where CLI is to store the column data or the LOB locator when the fetch occurs.</p> <p>This buffer is used to return data when any of the following functions are called: <code>SQLFetch()</code>, <code>SQLFetchScroll()</code>, <code>SQLSetPos()</code> using the <i>Operation</i> argument <code>SQL_REFRESH</code>, or <code>SQLBulkOperations()</code> using the <i>Operation</i> argument <code>SQL_FETCH_BY_BOOKMARK</code>. Otherwise, <code>SQLBulkOperations()</code> and <code>SQLSetPos()</code> use the buffer to retrieve data.</p> <p>If <i>TargetValuePtr</i> is null, the column is unbound. All columns can be unbound with a call to <code>SQLFreeStmt()</code> with the <code>SQL_UNBIND</code> option.</p>
SQLLEN	<i>BufferLength</i>	input	<p>Size in bytes of <i>TargetValuePtr</i> buffer available to store the column data or the LOB locator.</p> <p>If <i>TargetType</i> denotes a binary or character string (either single or double byte) or is <code>SQL_C_DEFAULT</code> for a column returning variable length data, then <i>BufferLength</i> must be &gt; 0, or an error will be returned. Note that for character data, the driver counts the NULL termination character and so space must be allocated for it. For all other data types, this argument is ignored.</p>
SQLLEN *	<i>StrLen_or_IndPtr</i>	input/output (deferred)	<p>Pointer to value (or array of values) which indicates the number of bytes CLI has available to return in the <i>TargetValuePtr</i> buffer. If <i>TargetType</i> is a LOB locator, the size of the locator is returned, not the size of the LOB data.</p> <p>This buffer is used to return data when any of the following functions are called: <code>SQLFetch()</code>, <code>SQLFetchScroll()</code>, <code>SQLSetPos()</code> using the <i>Operation</i> argument <code>SQL_REFRESH</code>, or <code>SQLBulkOperations()</code> using the <i>Operation</i> argument <code>SQL_FETCH_BY_BOOKMARK</code>. Otherwise, <code>SQLBulkOperations()</code> and <code>SQLSetPos()</code> use the buffer to retrieve data.</p> <p><code>SQLFetch()</code> returns <code>SQL_NULL_DATA</code> in this argument if the data value of the column is null.</p> <p>This pointer value must be unique for each bound column, or NULL.</p> <p>A value of <code>SQL_COLUMN_IGNORE</code>, <code>SQL_NTS</code>, <code>SQL_NULL_DATA</code>, or the length of the data can be set for use with <code>SQLBulkOperations()</code>.</p> <p><code>SQL_NO_LENGTH</code> might also be returned, refer to the Usage section below for more information.</p>

- For this function, both *TargetValuePtr* and *StrLen\_or\_IndPtr* are deferred outputs, meaning that the storage locations these pointers point to do not get updated



## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

until a result set row is fetched. As a result, the locations referenced by these pointers must remain valid until `SQLFetch()` or `SQLFetchScroll()` is called. For example, if `SQLBindCol()` is called within a local function, `SQLFetch()` must be called from within the same scope of the function or the `TargetValuePtr` buffer must be allocated as static or global.

- CLI will be able to optimize data retrieval for all variable length data types if `TargetValuePtr` is placed consecutively in memory after `StrLen_or_IndPtr`.

### Usage

Call `SQLBindCol()` once for each column in the result set for which either the data or, for LOB columns, the LOB locator is to be retrieved. When `SQLFetch()` or `SQLFetchScroll()` is called to retrieve data from the result set, the data in each of the bound columns is placed in the locations assigned by the `TargetValuePtr` and `StrLen_or_IndPtr` pointers. When the statement attribute `SQL_ATTR_ROW_ARRAY_SIZE` is greater than 1, then `TargetType` should refer to an array of buffers. If `TargetType` is a LOB locator, a locator value is returned, not the actual LOB data. The LOB locator references the entire data value in the LOB column.

If a CLI application does not provide an output buffer for a LOB column using the function `SQLBindCol()` the IBM data server client will, by default, request a LOB locator on behalf of the application for each LOB column in the result sets.

Columns are identified by a number, assigned sequentially from left to right.

- Column numbers start at 1 if bookmarks are not used (`SQL_ATTR_USE_BOOKMARKS` statement attribute set to `SQL_UB_OFF`).
- Column numbers start at 0 if bookmarks are used (the statement attribute set to `SQL_UB_ON`).

After columns have been bound, in subsequent fetches the application can change the binding of these columns or bind previously unbound columns by calling `SQLBindCol()`. The new binding does not apply to data already fetched, it will be used on the next fetch. To unbind a single column (including columns bound with `SQLBindFileToCol()`), call `SQLBindCol()` with the `TargetValuePtr` pointer set to `NULL`. To unbind all the columns, the application should call `SQLFreeStmt()` with the `Option` input set to `SQL_UNBIND`.

The application must ensure enough storage is allocated for the data to be retrieved. If the buffer is to contain variable length data, the application must allocate as much storage as the maximum length of the bound column plus the `NULL` terminator. Otherwise, the data might be truncated. If the buffer is to contain fixed length data, CLI assumes the size of the buffer is the length of the C data type. If data conversion is specified, the required size might be affected.

If string truncation does occur, `SQL_SUCCESS_WITH_INFO` is returned and `StrLen_or_IndPtr` will be set to the actual size of `TargetValuePtr` available for return to the application.

Truncation is also affected by the `SQL_ATTR_MAX_LENGTH` statement attribute (used to limit the amount of data returned to the application). The application can specify not to report truncation by calling `SQLSetStmtAttr()` with `SQL_ATTR_MAX_LENGTH` and a value for the maximum length to return for all variable length columns, and by allocating a `TargetValuePtr` buffer of the same size (plus the null-terminator). If the column data is larger than the set maximum

## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

length, SQL\_SUCCESS will be returned when the value is fetched and the maximum length, not the actual length, will be returned in *StrLen\_or\_IndPtr*.

If the column to be bound is a SQL\_GRAPHIC, SQL\_VARGRAPHIC or SQL\_LONGVARGRAPHIC type, then *TargetType* can be set to SQL\_C\_DBCHAR or SQL\_C\_CHAR. If *TargetType* is SQL\_C\_DBCHAR, the data fetched into the *TargetValuePtr* buffer will be null-terminated with a double byte null-terminator. If *TargetType* is SQL\_C\_CHAR, then there will be no null-termination of the data. In both cases, the length of the *TargetValuePtr* buffer (*BufferLength*) is in units of bytes and should therefore be a multiple of 2. It is also possible to force CLI to null terminate graphic strings using the PATCH1 keyword.

**Note:** SQL\_NO\_TOTAL will be returned in *StrLen\_or\_IndPtr* if:

- The SQL type is a variable length type, and
- *StrLen\_or\_IndPtr* and *TargetValuePtr* are contiguous, and
- The column type is NOT NULLABLE, and
- String truncation occurred.

### Descriptors and SQLBindCol

The following sections describe how SQLBindCol() interacts with descriptors.

**Note:** Calling SQLBindCol() for one statement can affect other statements. This occurs when the ARD associated with the statement is explicitly allocated and is also associated with other statements. Because SQLBindCol() modifies the descriptor, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate this descriptor from the other statements before calling SQLBindCol().

#### Argument mappings

Conceptually, SQLBindCol() performs the following steps in sequence:

- Calls SQLGetStmtAttr() to obtain the ARD handle.
- Calls SQLGetDescField() to get this descriptor's SQL\_DESC\_COUNT field, and if the value in the *ColumnNumber* argument exceeds the value of SQL\_DESC\_COUNT, calls SQLSetDescField() to increase the value of SQL\_DESC\_COUNT to *ColumnNumber*.
- Calls SQLSetDescField() multiple times to assign values to the following fields of the ARD:
  - Sets SQL\_DESC\_TYPE and SQL\_DESC\_CONCISE\_TYPE to the value of *TargetType*.
  - Sets one or more of SQL\_DESC\_LENGTH, SQL\_DESC\_PRECISION, SQL\_DESC\_SCALE as appropriate for *TargetType*.
  - Sets the SQL\_DESC\_OCTET\_LENGTH field to the value of *BufferLength*.
  - Sets the SQL\_DESC\_DATA\_PTR field to the value of *TargetValue*.
  - Sets the SQL\_DESC\_INDICATOR\_PTR field to the value of *StrLen\_or\_IndPtr* (see the following paragraph).
  - Sets the SQL\_DESC\_OCTET\_LENGTH\_PTR field to the value of *StrLen\_or\_IndPtr* (see the following paragraph).

The variable that the *StrLen\_or\_IndPtr* argument refers to is used for both indicator and length information. If a fetch encounters a null value for the column, it stores SQL\_NULL\_DATA in this variable; otherwise, it stores the data length in this variable. Passing a null pointer as *StrLen\_or\_IndPtr* keeps the fetch operation from returning the data length, but makes the fetch fail if it encounters a null value and has no way to return SQL\_NULL\_DATA.

## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

If the call to `SQLBindCol()` fails, the content of the descriptor fields it would have set in the ARD are undefined, and the value of the `SQL_DESC_COUNT` field of the ARD is unchanged.

Implicit resetting of `COUNT` field

`SQLBindCol()` sets `SQL_DESC_COUNT` to the value of the *ColumnNumber* argument only when this would increase the value of `SQL_DESC_COUNT`. If the value in the *TargetValuePtr* argument is a null pointer and the value in the *ColumnNumber* argument is equal to `SQL_DESC_COUNT` (that is, when unbinding the highest bound column), then `SQL_DESC_COUNT` is set to the number of the highest remaining bound column.

Cautions regarding `SQL_C_DEFAULT`

To retrieve column data successfully, the application must determine correctly the length and starting point of the data in the application buffer. When the application specifies an explicit *TargetType*, application misconceptions are readily detected. However, when the application specifies a *TargetType* of `SQL_C_DEFAULT`, `SQLBindCol()` can be applied to a column of a different data type from the one intended by the application, either from changes to the metadata or by applying the code to a different column. In this case, the application might fail to determine the start or length of the fetched column data. This can lead to unreported data errors or memory violations.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 5. *SQLBindCol* SQLSTATES

SQLSTATE	Description	Explanation
07009	Invalid descriptor index	The value specified for the argument <i>ColumnNumber</i> exceeded the maximum number of columns in the result set, or the value specified was less than 0.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	<i>TargetType</i> was not a valid data type or <code>SQL_C_DEFAULT</code> .
HY010	Function sequence error.	The function was called while in a data-at-execute ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.  The function was called while within a <code>BEGIN COMPOUND</code> and <code>END COMPOUND</code> SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

## SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

Table 5. SQLBindCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The value specified for the argument <i>BufferLength</i> is less than 1 and the argument <i>TargetType</i> is either SQL_C_CHAR, SQL_C_BINARY or SQL_C_DEFAULT.
HYC00	Driver not capable.	CLI recognizes, but does not support the data type specified in the argument <i>TargetType</i>  A LOB locator C data type was specified, but the connected server does not support LOB data types.

**Note:** Additional diagnostic messages relating to the bound columns might be reported at fetch time.

### Restrictions

The LOB data support is only available when connected to a server that supports large object data types. If the application attempts to specify a LOB locator C data type for a server that does not support it, SQLSTATE HYC00 will be returned.

### Example

```
/* bind column 1 to variable */  
cliRC = SQLBindCol(hstmt, 1, SQL_C_SHORT, &deptnumb.val, 0, &deptnumb.ind);
```

## SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column

### Purpose

Specification:	CLI 2.1		
----------------	---------	--	--

SQLBindFileToCol() is used to associate or bind a LOB or XML column in a result set to a file reference or an array of file references. This enables data in that column to be transferred directly into a file when each row is fetched for the statement handle.

The LOB file reference arguments (file name, file name length, file reference options) refer to a file within the application's environment (on the client). Before fetching each row, the application must make sure that these variables contain the name of a file, the length of the file name, and a file option (new / overwrite / append). These values can be changed between each row fetch operation.

### Syntax

```
SQLRETURN SQLBindFileToCol (SQLHSTMT          StatementHandle, /* hstmt */  
                             SQLUSMALLINT      ColumnNumber,   /* icol */  
                             SQLCHAR           *FileName,  
                             SQLSMALLINT       *FileNameLength,  
                             SQLUINTEGER       *FileOptions,  
                             SQLSMALLINT       MaxFileNameLength,  
                             SQLINTEGER        *StringLength,  
                             SQLINTEGER        *IndicatorValue);
```

## SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column

### Function arguments

Table 6. SQLBindFileToCol arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLUSMALLINT	<i>icol</i>	input	Number identifying the column. Columns are numbered sequentially, from left to right, starting at 1.
SQLCHAR *	<i>FileName</i>	input (deferred)	Pointer to the location that will contain the file name or an array of file names at the time of the next fetch using the <i>StatementHandle</i> . This is either the complete path name of the file(s) or a relative file name(s). If relative file name(s) are provided, they are appended to the current path of the running application. This pointer cannot be NULL.
SQLSMALLINT *	<i>FileNameLength</i>	input (deferred)	Pointer to the location that will contain the length of the file name (or an array of lengths) at the time of the next fetch using the <i>StatementHandle</i> . If this pointer is NULL, then the <i>FileName</i> will be considered a null-terminated string, similar to passing a length of SQL_NTS.  The maximum value of the file name length is 255.
SQLINTEGER *	<i>FileOptions</i>	input (deferred)	Pointer to the location that will contain the file option or (array of file options) to be used when writing the file at the time of the next fetch using the <i>StatementHandle</i> . The following <i>FileOptions</i> are supported:  <b>SQL_FILE_CREATE</b> Create a new file. If a file by this name already exists, SQL_ERROR will be returned.  <b>SQL_FILE_OVERWRITE</b> If the file already exists, overwrite it. Otherwise, create a new file.  <b>SQL_FILE_APPEND</b> If the file already exists, append the data to it. Otherwise, create a new file.  Only one option can be chosen per file, there is no default.
SQLSMALLINT	<i>MaxFileNameLength</i>	input	This specifies the length of the <i>FileName</i> buffer or, if the application uses SQLFetchScroll() to retrieve multiple rows for the LOB column, this specifies the length of each element in the <i>FileName</i> array.
SQLINTEGER *	<i>StringLength</i>	output (deferred)	Pointer to the location that contains the length (or array of lengths) in bytes of the LOB data that is returned. If this pointer is NULL, nothing is returned.
SQLINTEGER *	<i>IndicatorValue</i>	output (deferred)	Pointer to the location that contains an indicator value (or array of values).

## SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column

### Usage

The application calls `SQLBindFileToCol()` once for each column that should be transferred directly to a file when a row is fetched. LOB data is written directly to the file without any data conversion, and without appending null-terminators. XML data is written out in UTF-8, with an XML declaration generated according to the setting of the `SQL_ATTR_XML_DECLARATION` connection or statement attribute.

*FileName*, *FileNameLength*, and *FileOptions* must be set before each fetch. When `SQLFetch()` or `SQLFetchScroll()` is called, the data for any column which has been bound to a LOB file reference is written to the file or files pointed to by that file reference. Errors associated with the deferred input argument values of `SQLBindFileToCol()` are reported at fetch time. The LOB file reference, and the deferred *StringLength* and *IndicatorValue* output arguments are updated between fetch operations.

If `SQLFetchScroll()` is used to retrieve multiple rows for the LOB column, *FileName*, *FileNameLength*, and *FileOptions* point to arrays of LOB file reference variables. In this case, *MaxFileNameLength* specifies the length of each element in the *FileName* array and is used by CLI to determine the location of each element in the *FileName* array. The contents of the array of file references must be valid at the time of the `SQLFetchScroll()` call. The *StringLength* and *IndicatorValue* pointers each point to an array whose elements are updated upon the `SQLFetchScroll()` call.

Using `SQLFetchScroll()`, multiple LOB values can be written to multiple files, or to the same file depending on the file names specified. If writing to the same file, the `SQL_FILE_APPEND` file option should be specified for each file name entry. Only column-wise binding of arrays of file references is supported with `SQLFetchScroll()`.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 7. *SQLBindFileToCol* SQLSTATEs

SQLSTATE	Description	Explanation
07009	Invalid column number.	The value specified for the argument <i>icol</i> was less than 1.  The value specified for the argument <i>icol</i> exceeded the maximum number of columns supported by the data source.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	<i>FileName</i> , <i>StringLength</i> or <i>FileOptions</i> is a null pointer.

## SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column

Table 7. SQLBindFileToCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument <i>MaxFileNameLength</i> was less than 0.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.

### Restrictions

This function is not available when connected to DB2 servers that do not support large object data types. Call `SQLGetFunctions()` with the function type set to `SQL_API_SQLBINDFILETOCOL` and check the *SupportedPtr* output argument to determine if the function is supported for the current connection.

### Example

```
/* bind a file to the BLOB column */
rc = SQLBindFileToCol(hstmt,
                      1,
                      fileName,
                      &fileNameLength,
                      &fileOption,
                      14,
                      NULL,
                      &fileInd);
```

## SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter

### Purpose

Specification:	CLI 2.1		
----------------	---------	--	--

`SQLBindFileToParam()` is used to associate or bind a parameter marker in an SQL statement to a file reference or an array of file references. This enables data from the file to be transferred directly into a LOB or XML column when the statement is subsequently executed.

The LOB file reference arguments (file name, file name length, file reference options) refer to a file within the application's environment (on the client). Before calling `SQLExecute()` or `SQLExecDirect()`, the application must make sure that this information is available in the deferred input buffers. These values can be changed between `SQLExecute()` calls.

### Syntax

```
SQLRETURN SQLBindFileToParam (
    SQLHSTMT      StatementHandle,      /* hstmt */
    SQLUSMALLINT  TargetType,           /* ipar */
```

## SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter

```

SQLSMALLINT      DataType,           /* fSqlType */
SQLCHAR          *FileName,
SQLSMALLINT      *FileNameLength,
SQLINTEGER       *FileOptions,
SQLSMALLINT      MaxFileNameLength,
SQLINTEGER       *IndicatorValue);
    
```

### Function arguments

Table 8. SQLBindFileToParam arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLSMALLINT	<i>TargetType</i>	input	Parameter marker number. Parameters are numbered sequentially, from left to right, starting at 1.
SQLSMALLINT	<i>DataType</i>	input	SQL Data Type of the column. The data type must be one of: <ul style="list-style-type: none"> <li>• SQL_BLOB</li> <li>• SQL_CLOB</li> <li>• SQL_DBCLOB</li> <li>• SQL_XML</li> </ul>
SQLCHAR *	<i>FileName</i>	input (deferred)	Pointer to the location that will contain the file name or an array of file names when the statement ( <i>StatementHandle</i> ) is executed. This is either the complete path name of the file or a relative file name. If a relative file name is provided, it is appended to the current path of the client process.  This argument cannot be NULL.
SQLSMALLINT *	<i>FileNameLength</i>	input (deferred)	Pointer to the location that will contain the length of the file name (or an array of lengths) at the time of the next <code>SQLExecute()</code> or <code>SQLExecDirect()</code> using the <i>StatementHandle</i> .  If this pointer is NULL, then the <i>FileName</i> will be considered a null-terminated string, similar to passing a length of <code>SQL_NTS</code> .  The maximum value of the file name length is 255.
SQLINTEGER *	<i>FileOptions</i>	input (deferred)	Pointer to the location that will contain the file option (or an array of file options) to be used when reading the file. The location will be accessed when the statement ( <i>StatementHandle</i> ) is executed. Only one option is supported (and it must be specified):  <b>SQL_FILE_READ</b> A regular file that can be opened, read and closed. (The length is computed when the file is opened)  This pointer cannot be NULL.
SQLSMALLINT	<i>MaxFileNameLength</i>	input	This specifies the length of the <i>FileName</i> buffer. If the application calls <code>SQLParamOptions()</code> to specify multiple values for each parameter, this is the length of each element in the <i>FileName</i> array.



## SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter

Table 8. SQLBindFileToParam arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	<i>IndicatorValue</i>	input (deferred)	Pointer to the location that contains an indicator value (or array of values), which is set to SQL_NULL_DATA if the data value of the parameter is to be null. It must be set to 0 (or the pointer can be set to null) when the data value is not null.

### Usage

The application calls SQLBindFileToParam() once for each parameter marker whose value should be obtained directly from a file when a statement is executed. Before the statement is executed, *FileName*, *FileNameLength*, and *FileOptions* values must be set. When the statement is executed, the data for any parameter which has been bound using SQLBindFileToParam() is read from the referenced file and passed to the server.

If the application uses SQLParamOptions() to specify multiple values for each parameter, then *FileName*, *FileNameLength*, and *FileOptions* point to an array of LOB file reference variables. In this case, *MaxFileNameLength* specifies the length of each element in the *FileName* array and is used by CLI to determine the location of each element in the *FileName* array.

A LOB parameter marker can be associated with (bound to) an input file using SQLBindFileToParam(), or with a stored buffer using SQLBindParameter(). The most recent bind parameter function call determines the type of binding that is in effect.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 9. SQLBindFileToParam SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY004	SQL data type out of range.	The value specified for <i>DataType</i> was not a valid SQL type for this function call.
HY009	Invalid argument value.	<i>FileName</i> , <i>FileOptions</i> <i>FileNameLength</i> , is a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.

## SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter

Table 9. SQLBindFileToParam SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the input argument <i>MaxFileNameLength</i> was less than 0.
HY093	Invalid parameter number.	The value specified for <i>TargetType</i> was either less than 1 or greater than the maximum number of parameters supported.
HYC00	Driver not capable.	The server does not support Large Object data types.

### Restrictions

This function is not available when connected to DB2 servers that do not support large object data types. Call `SQLGetFunctions()` with the function type set to `SQL_API_SQLBINDFILETOPARAM` and check the *SupportedPtr* output argument to determine if the function is supported for the current connection.

### Example

```
/* bind the file parameter */
rc = SQLBindFileToParam(hstmt,
                        3,
                        SQL_BLOB,
                        fileName,
                        &fileNameLength,
                        &fileOption,
                        14,
                        &fileInd);
```

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

### Purpose

Specification:	CLI 2.1	ODBC 2.0	
----------------	---------	----------	--

`SQLBindParameter()` is used to associate or bind parameter markers in an SQL statement to either:

- Application variables or arrays of application variables (storage buffers) for all C data types. In this case data is transferred from the application to the DBMS when `SQLExecute()` or `SQLExecDirect()` is called. Data conversion might occur as the data is transferred.
- A LOB locator, for SQL LOB data types. In this case a LOB locator value, not the LOB data itself, is transferred from the application to the server when the SQL statement is executed.

Alternatively, LOB parameters can be bound directly to a file using `SQLBindFileToParam()`

This function must also be used to bind a parameter of a stored procedure CALL statement to the application where the parameter can be input, output or both.

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

### Syntax

```
SQLRETURN SQLBindParameter(
    SQLHSTMT      StatementHandle, /* hstmt */
    SQLUSMALLINT  ParameterNumber, /* ipar */
    SQLSMALLINT   InputOutputType, /* fParamType */
    SQLSMALLINT   ValueType,       /* fCType */
    SQLSMALLINT   ParameterType,   /* fSqlType */
    SQLULEN       ColumnSize,      /* cbColDef */
    SQLSMALLINT   DecimalDigits,   /* ibScale */
    SQLPOINTER    ParameterValuePtr, /* rgbValue */
    SQLLEN        BufferLength,     /* cbValueMax */
    SQLLEN        *StrLen_or_IndPtr); /* pcbValue */
```

### Function arguments

Table 10. SQLBindParameter arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement Handle
SQLUSMALLINT	<i>ParameterNumber</i>	input	Parameter marker number, ordered sequentially left to right, starting at 1.
SQLSMALLINT	<i>InputOutputType</i>	input	<p>The type of parameter. The value of the SQL_DESC_PARAMETER_TYPE field of the IPD is also set to this argument. The supported types are:</p> <ul style="list-style-type: none"> <li>SQL_PARAM_INPUT: The parameter marker is associated with an SQL statement that is not a stored procedure CALL; or, it marks an input parameter of the CALLED stored procedure. When the statement is executed, the data for the parameter is sent to the server and as such, the <i>ParameterValuePtr</i> buffer must contain valid input data value(s), unless the <i>StrLen_or_IndPtr</i> buffer contains SQL_NULL_DATA or SQL_DATA_AT_EXEC (if the value should be sent via SQLParamData() and SQLPutData()).</li> <li>SQL_PARAM_INPUT_OUTPUT: The parameter marker is associated with an input/output parameter of the CALLED stored procedure. When the statement is executed, the data for the parameter is sent to the server and as such, the <i>ParameterValuePtr</i> buffer must contain valid input data value(s), unless the <i>StrLen_or_IndPtr</i> buffer contains SQL_NULL_DATA or SQL_DATA_AT_EXEC (if the value should be sent via SQLParamData() and SQLPutData()).</li> <li>SQL_PARAM_OUTPUT: The parameter marker is associated with an output parameter of the CALLED stored procedure or the return value of the stored procedure. After the statement is executed, data for the output parameter is returned to the application buffer specified by <i>ParameterValuePtr</i> and <i>StrLen_or_IndPtr</i>, unless both are NULL pointers, in which case the output data is discarded. If an output parameter does not have a return value then <i>StrLen_or_IndPtr</i> is set to SQL_NULL_DATA.</li> </ul>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>ValueType</i>	input	<p>C data type of the parameter. The following types are supported:</p> <ul style="list-style-type: none"> <li>• SQL_C_BINARY</li> <li>• SQL_C_BIT</li> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CHAR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCHAR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> <li>• SQL_C_DECIMAL_IBM</li> <li>• SQL_C_DOUBLE</li> <li>• SQL_C_FLOAT</li> <li>• SQL_C_LONG</li> <li>• SQL_C_NUMERIC<sup>a</sup></li> <li>• SQL_C_SBIGINT</li> <li>• SQL_C_SHORT</li> <li>• SQL_C_TYPE_DATE</li> <li>• SQL_C_TYPE_TIME</li> <li>• SQL_C_TYPE_TIMESTAMP</li> <li>• SQL_C_TYPE_TIMESTAMP_EXT</li> <li>• SQL_C_TYPE_TIMESTAMP_EXT_TZ</li> <li>• SQL_C_TINYINT</li> <li>• SQL_C_UBIGINT</li> <li>• SQL_C_UTINYINT</li> <li>• SQL_C_WCHAR</li> </ul> <p>Specifying SQL_C_DEFAULT causes data to be transferred from its default C data type to the type indicated in <i>ParameterType</i>.</p> <p><sup>a</sup> Windows 32-bit only</p>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>ParameterType</i>	input	<p>SQL data type of the parameter. The supported types are:</p> <ul style="list-style-type: none"> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BIT</li> <li>• SQL_BLOB</li> <li>• SQL_BLOB_LOCATOR</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_CLOB_LOCATOR</li> <li>• SQL_DBCLOB</li> <li>• SQL_DBCLOB_LOCATOR</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_LONGVARIABLE</li> <li>• SQL_LONGVARIABLE</li> <li>• SQL_LONGVARIABLE</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TINYINT</li> <li>• SQL_TYPE_DATE</li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_TYPE_TIMESTAMP_WITH_TIMEZONE</li> <li>• SQL_VARIABLE</li> <li>• SQL_VARIABLE</li> <li>• SQL_VARIABLE</li> <li>• SQL_WCHAR</li> <li>• SQL_XML</li> </ul> <p><b>Note:</b> SQL_BLOB_LOCATOR, SQL_CLOB_LOCATOR, SQL_DBCLOB_LOCATOR are application related concepts and do not map to a data type for column definition during a CREATE TABLE statement.</p>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLULEN	<i>ColumnSize</i>	input	<p>Precision of the corresponding parameter marker. If <i>ParameterType</i> denotes:</p> <ul style="list-style-type: none"> <li>• A binary or single byte character string (for example, SQL_CHAR, SQL_BLOB), this is the maximum length in bytes for this parameter marker.</li> <li>• A double byte character string (for example, SQL_GRAPHIC), this is the maximum length in double-byte characters for this parameter.</li> <li>• SQL_DECIMAL, SQL_NUMERIC, this is the maximum decimal precision.</li> <li>• An XML value (SQL_XML) for an external routine argument, this is the maximum length in bytes, n, of the declared XML AS CLOB(n) argument. For all other parameters of type SQL_XML, this argument is ignored.</li> <li>• Otherwise, this argument is ignored.</li> </ul>
SQLSMALLINT	<i>DecimalDigits</i>	input	<p>If <i>ParameterType</i> is SQL_DECIMAL or SQL_NUMERIC, <i>DecimalDigits</i> represents the scale of the corresponding parameter and sets the SQL_DESC_SCALE field of the IPD.</p> <p>If <i>ParameterType</i> is SQL_TYPE_TIMESTAMP or SQL_TYPE_TIME, <i>Decimal Digits</i> represents the precision of the corresponding parameter and sets the SQL_DESC_PRECISION field of the IPD. The precision of a time timestamp value is the number of digits to the right of the decimal point in the string representation of a time or timestamp (for example, the scale of yyyy-mm-dd hh:mm:ss.fff is 3).</p> <p>Other than for the <i>ParameterType</i> values mentioned here, <i>DecimalDigits</i> is ignored.</p>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	<i>ParameterValuePtr</i>	input (deferred), output (deferred), or both	<ul style="list-style-type: none"> <li>On input (<i>InputOutputType</i> set to SQL_PARAM_INPUT, or SQL_PARAM_INPUT_OUTPUT): At execution time, if <i>StrLen_or_IndPtr</i> does not contain SQL_NULL_DATA or SQL_DATA_AT_EXEC, then <i>ParameterValuePtr</i> points to a buffer that contains the actual data for the parameter. If <i>StrLen_or_IndPtr</i> contains SQL_DATA_AT_EXEC, then <i>ParameterValuePtr</i> is an application-defined 32-bit value that is associated with this parameter. This 32-bit value is returned to the application via a subsequent SQLParamData() call. If SQLParamOptions() is called to specify multiple values for the parameter, then <i>ParameterValuePtr</i> is a pointer to a input buffer array of <i>BufferLength</i> bytes.</li> <li>On output (<i>InputOutputType</i> set to SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT): <i>ParameterValuePtr</i> points to the buffer where the output parameter value of the stored procedure will be stored. If <i>InputOutputType</i> is set to SQL_PARAM_OUTPUT, and both <i>ParameterValuePtr</i> and <i>StrLen_or_IndPtr</i> are NULL pointers, then the output parameter value or the return value from the stored procedure call is discarded.</li> </ul>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLLEN	<i>BufferLength</i>	input	<p>For character and binary data, <i>BufferLength</i> specifies the length of the <i>ParameterValuePtr</i> buffer (if is treated as a single element) or the length of each element in the <i>ParameterValuePtr</i> array (if the application calls <code>SQLParamOptions()</code> to specify multiple values for each parameter). For non-character and non-binary data, this argument is ignored -- the length of the <i>ParameterValuePtr</i> buffer (if it is a single element) or the length of each element in the <i>ParameterValuePtr</i> array (if <code>SQLParamOptions()</code> is used to specify an array of values for each parameter) is assumed to be the length associated with the C data type.</p> <p>For output parameters, <i>BufferLength</i> is used to determine whether to truncate character or binary output data in the following manner:</p> <ul style="list-style-type: none"> <li>• For character data, if the number of bytes available to return is greater than or equal to <i>BufferLength</i>, the data in <i>ParameterValuePtr</i> is truncated to <i>BufferLength-1</i> bytes and is null-terminated (unless null-termination has been turned off).</li> <li>• For binary data, if the number of bytes available to return is greater than <i>BufferLength</i>, the data in <i>ParameterValuePtr</i> is truncated to <i>BufferLength</i> bytes.</li> </ul>



## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLLEN *	<i>StrLen_or_IndPtr</i>	input (deferred), output (deferred), or both	<p>If this is an input or input/output parameter:</p> <p>This is the pointer to the location which contains (when the statement is executed) the length of the parameter marker value stored at <i>ParameterValuePtr</i>.</p> <p>To specify a null value for a parameter marker, this storage location must contain SQL_NULL_DATA.</p> <p>If <i>ValueType</i> is SQL_C_CHAR, this storage location must contain either the exact length of the data stored at <i>ParameterValuePtr</i>, or SQL_NTS if the contents at <i>ParameterValuePtr</i> is null-terminated.</p> <p>If <i>ValueType</i> indicates character data (explicitly, or implicitly using SQL_C_DEFAULT), and this pointer is set to NULL, it is assumed that the application will always provide a null-terminated string in <i>ParameterValuePtr</i>. This also implies that this parameter marker will never have a null value.</p> <p>If <i>ParameterType</i> denotes a graphic data type and the <i>ValueType</i> is SQL_C_CHAR, the pointer to <i>StrLen_or_IndPtr</i> can never be NULL and the contents of <i>StrLen_or_IndPtr</i> can never hold SQL_NTS. In general for graphic data types, this length should be the number of octets that the double byte data occupies; therefore, the length should always be a multiple of 2. In fact, if the length is odd, then an error will occur when the statement is executed.</p> <p>When SQLExecute() or SQLExecDirect() is called, and <i>StrLen_or_IndPtr</i> points to a value of SQL_DATA_AT_EXEC, the data for the parameter will be sent with SQLPutData(). This parameter is referred to as a <b>data-at-execution</b> parameter. When SQLBindParameter() or SQLExtendedBind() method is called through after setting the SQL_ATTR_EXTENDED_INDICATORS attribute, the <i>StrLen_or_IndPtr</i> argument allows SQL_UNASSIGNED and SQL_DEFAULT_PARAM constant to pass through the method.</p>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	StrLen_or_IndPtr (cont)	input (deferred), output (deferred), or both	<p>If SQLSetStmtAttr() is used with the SQL_ATTR_PARAMSET_SIZE attribute to specify multiple values for each parameter, StrLen_or_IndPtr points to an array of SQLINTEGER values where each of the elements can be the number of bytes in the corresponding ParameterValuePtr element (excluding the null-terminator), or SQL_NULL_DATA.</p> <p>The StrLen_or_IndPtr represents the size of the parameter. If you have an output parameter, StrLen_or_IndPtr is a memory address (a pointer) to an SQLINTEGER and the value will contain either:</p> <ul style="list-style-type: none"><li>• The length of the buffer (minus the NULL terminator).</li><li>• -1 (SQL_NULL_DATA), which means that the value is NULL, and you can ignore the actual value.</li><li>• -4 (SQL_NO_TOTAL), which is only used for LOB type of data, and is used to indicate that the number of bytes available to return cannot be determined.</li></ul>

### Usage

SQLBindParameter() extends the capability of the deprecated SQLSetParam() function, by providing a method of:

- Specifying whether a parameter is input, input / output, or output, necessary for proper handling of parameters for stored procedures.
- Specifying an array of input parameter values when SQLSetStmtAttr() with the SQL\_ATTR\_PARAMSET\_SIZE attribute is used in conjunction with SQLBindParameter().

This function can be called before SQLPrepare() if the data types and lengths of the target columns in the WHERE or UPDATE clause, or the parameters for the stored procedure are known. Otherwise, you can obtain the attributes of the target columns or stored procedure parameters after the statement is prepared using SQLDescribeParam(), and then bind the parameter markers.

Parameter markers are referenced by number (*ParameterNumber*) and are numbered sequentially from left to right, starting at 1.

The C buffer data type given by *ValueType* must be compatible with the SQL data type indicated by *ParameterType*, or an error will occur.

All parameters bound by this function remain in effect until one of the following occurs:

- SQLFreeStmt() is called with the SQL\_RESET\_PARAMS option, or
- SQLFreeHandle() is called with *HandleType* set to SQL\_HANDLE\_STMT, or SQLFreeStmt() is called with the SQL\_DROP option, or
- SQLBindParameter() is called again for the same *ParameterNumber*, or

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

- `SQLSetDescField()` is called, with the associated APD descriptor handle, to set `SQL_DESC_COUNT` in the header field of the APD to zero (0).

A parameter can only be bound to either a file or a storage location, not both. The most recent parameter binding function call determines the bind that is in effect.

### Parameter type

The *InputOutputType* argument specifies the type of the parameter. All parameters in the SQL statements that do not call procedures are input parameters. Parameters in stored procedure calls can be input, input/output, or output parameters. Even though the DB2 stored procedure argument convention typically implies that all procedure arguments are input/output, the application programmer can still choose to specify more exactly the input or output nature on the `SQLBindParameter()` to follow a more rigorous coding style.

- If an application cannot determine the type of a parameter in a procedure call, set *InputOutputType* to `SQL_PARAM_INPUT`; if the data source returns a value for the parameter, CLI discards it.
- If an application has marked a parameter as `SQL_PARAM_INPUT_OUTPUT` or `SQL_PARAM_OUTPUT` and the data source does not return a value, CLI sets the *StrLen\_or\_IndPtr* buffer to `SQL_NULL_DATA`.
- If an application marks a parameter as `SQL_PARAM_OUTPUT`, data for the parameter is returned to the application after the `CALL` statement has been processed. If the *ParameterValuePtr* and *StrLen\_or\_IndPtr* arguments are both null pointers, CLI discards the output value. If the data source does not return a value for an output parameter, CLI sets the *StrLen\_or\_IndPtr* buffer to `SQL_NULL_DATA`.
- For this function, *ParameterValuePtr* and *StrLen\_or\_IndPtr* are deferred arguments. In the case where *InputOutputType* is set to `SQL_PARAM_INPUT` or `SQL_PARAM_INPUT_OUTPUT`, the storage locations must be valid and contain input data values when the statement is executed. This means either keeping the `SQLExecDirect()` or `SQLExecute()` call in the same procedure scope as the `SQLBindParameter()` calls, or, these storage locations must be dynamically allocated or statically / globally declared.

Similarly, if *InputOutputType* is set to `SQL_PARAM_OUTPUT` or `SQL_PARAM_INPUT_OUTPUT`, the *ParameterValuePtr* and *StrLen\_or\_IndPtr* buffer locations must remain valid until the `CALL` statement has been executed.

### ParameterValuePtr and StrLen\_or\_IndPtr arguments

*ParameterValuePtr* and *StrLen\_or\_IndPtr* are deferred arguments, so the storage locations they point to must be valid and contain input data values when the statement is executed. This means either keeping the `SQLExecDirect()` or `SQLExecute()` call in the same application function scope as the `SQLBindParameter()` calls, or dynamically allocating or statically or globally declaring these storage locations.

Since the data in the variables referenced by *ParameterValuePtr* and *StrLen\_or\_IndPtr* is not verified until the statement is executed, data content or format errors are not detected or reported until `SQLExecute()` or `SQLExecDirect()` is called.

An application can pass the value for a parameter either in the *ParameterValuePtr* buffer or with one or more calls to `SQLPutData()`. In the latter case, these parameters are data-at-execution parameters. The application informs CLI of a

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

data-at-execution parameter by placing the `SQL_DATA_AT_EXEC` value in the buffer pointed to by `StrLen_or_IndPtr`. It sets the `ParameterValuePtr` input argument to a 32-bit value which will be returned on a subsequent `SQLParamData()` call and can be used to identify the parameter position.

When `SQLBindParameter()` is used to bind an application variable to an output parameter for a stored procedure, CLI can provide some performance enhancement if the `ParameterValuePtr` buffer is placed consecutively in memory after the `StrLen_or_IndPtr` buffer. For example:

```
struct {  SQLINTEGER  StrLen_or_IndPtr;
         SQLCHAR    ParameterValuePtr[MAX_BUFFER];
        } column;
```

### BufferLength argument

For character and binary C data, the `BufferLength` argument specifies the length of the `ParameterValuePtr` buffer if it is a single element; or, if the application calls `SQLSetStmtAttr()` with the `SQL_ATTR_PARAMSET_SIZE` attribute to specify multiple values for each parameter, `BufferLength` is the length of *each* element in the `ParameterValuePtr` array, including the null-terminator. If the application specifies multiple values, `BufferLength` is used to determine the location of values in the `ParameterValuePtr` array. For all other types of C data, the `BufferLength` argument is ignored.

### ColumnSize argument

When actual size of the target column or output parameter is not known, the application can specify 0 for the length of the column. (`ColumnSize` set to 0).

If the column's data type is of fixed-length, the CLI driver will base the length from the data type itself. However, setting `ColumnSize` to 0 means different things when the data type is of type character, binary string or large object:

#### Input parameter

A 0 `ColumnSize` means that CLI will use the maximum length for the SQL type provided as the size of the column or stored procedure parameter. CLI will perform any necessary conversions using this size.

#### Output parameter (stored procedures only)

A 0 `ColumnSize` means that CLI will use `BufferLength` as the parameter's size. Note that this means that the stored procedure must not return more than `BufferLength` bytes of data or a truncation error will occur.

#### For Input-output parameter (store procedures only)

A 0 `ColumnSize` means that CLI will set both the input and output to `BufferLength` as the target parameter. This means that the input data will be converted to this new size if necessary before being sent to the stored procedure and at most `BufferLength` bytes of data are expected to be returned.

Setting `ColumnSize` to 0 is not recommended unless it is required; it causes CLI to perform costly checking for the length of the data at run time.

### Descriptors

How a parameter is bound is determined by fields of the APD and IPD. The arguments in `SQLBindParameter()` are used to set those descriptor fields. The fields

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

can also be set by the `SQLSetDescField()` functions, although `SQLBindParameter()` is more efficient to use because the application does not have to obtain a descriptor handle to call `SQLBindParameter()`.

**Note:** Calling `SQLBindParameter()` for one statement can affect other statements. This occurs when the APD associated with the statement is explicitly allocated and is also associated with other statements. Because `SQLBindParameter()` modifies the fields of the APD, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate the descriptor from the other statements before calling `SQLBindParameter()`.

Conceptually, `SQLBindParameter()` performs the following steps in sequence:

- Calls `SQLGetStmtAttr()` to obtain the APD handle.
- Calls `SQLGetDescField()` to get the `SQL_DESC_COUNT` header field from the APD, and if the value of the *ParameterNumber* argument exceeds the value of `SQL_DESC_COUNT`, calls `SQLSetDescField()` to increase the value of `SQL_DESC_COUNT` to *ParameterNumber*.
- Calls `SQLSetDescField()` multiple times to assign values to the following fields of the APD:
  - Sets `SQL_DESC_TYPE` and `SQL_DESC_CONCISE_TYPE` to the value of *ValueType*, except that if *ValueType* is one of the concise identifiers of a datetime, it sets `SQL_DESC_TYPE` to `SQL_DATETIME`, sets `SQL_DESC_CONCISE_TYPE` to the concise identifier, and sets `SQL_DESC_DATETIME_INTERVAL_CODE` to the corresponding datetime subcode.
  - Sets the `SQL_DESC_DATA_PTR` field to the value of *ParameterValue*.
  - Sets the `SQL_DESC_OCTET_LENGTH_PTR` field to the value of *StrLen\_or\_Ind*.
  - Sets the `SQL_DESC_INDICATOR_PTR` field also to the value of *StrLen\_or\_Ind*.

The *StrLen\_or\_Ind* parameter specifies both the indicator information and the length for the parameter value.

- Calls `SQLGetStmtAttr()` to obtain the IPD handle.
- Calls `SQLGetDescField()` to get the IPD's `SQL_DESC_COUNT` field, and if the value of the *ParameterNumber* argument exceeds the value of `SQL_DESC_COUNT`, calls `SQLSetDescField()` to increase the value of `SQL_DESC_COUNT` to *ParameterNumber*.
- Calls `SQLSetDescField()` multiple times to assign values to the following fields of the IPD:
  - Sets `SQL_DESC_TYPE` and `SQL_DESC_CONCISE_TYPE` to the value of *ParameterType*, except that if *ParameterType* is one of the concise identifiers of a datetime, it sets `SQL_DESC_TYPE` to `SQL_DATETIME`, sets `SQL_DESC_CONCISE_TYPE` to the concise identifier, and sets `SQL_DESC_DATETIME_INTERVAL_CODE` to the corresponding datetime subcode.
  - Sets one or more of `SQL_DESC_LENGTH`, `SQL_DESC_PRECISION`, and `SQL_DESC_SCALE` as appropriate for *ParameterType*.

If the call to `SQLBindParameter()` fails, the content of the descriptor fields that it would have set in the APD are undefined, and the `SQL_DESC_COUNT` field of the APD is unchanged. In addition, the `SQL_DESC_LENGTH`, `SQL_DESC_PRECISION`, `SQL_DESC_SCALE`, and `SQL_DESC_TYPE` fields of the appropriate record in the IPD are undefined and the `SQL_DESC_COUNT` field of the IPD is unchanged.

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 11. SQLBindParameter SQLSTATES

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The conversion from the data value identified by the <i>ValueType</i> argument to the data type identified by the <i>ParameterType</i> argument is not a meaningful conversion. (For example, conversion from SQL_C_TYPE_DATE to SQL_DOUBLE.)
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	The value specified by the argument <i>ParameterNumber</i> not a valid data type or SQL_C_DEFAULT.
HY004	SQL data type out of range.	The value specified for the argument <i>ParameterType</i> is not a valid SQL data type.
HY009	Invalid argument value.	The argument <i>ParameterValuePtr</i> was a null pointer and the argument <i>StrLen_or_IndPtr</i> was a null pointer, and <i>InputOutputType</i> is not SQL_PARAM_OUTPUT.
HY010	Function sequence error.	Function was called after SQLExecute() or SQLExecDirect() had returned SQL_NEED_DATA, but data has not been sent for all <i>data-at-execution</i> parameters.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check was not consistent.
HY090	Invalid string or buffer length.	The value specified for the argument <i>BufferLength</i> was less than 0.
HY093	Invalid parameter number.	The value specified for the argument <i>ValueType</i> was less than 1 or greater than the maximum number of parameters supported by the server.
HY094	Invalid scale value.	<p>The value specified for <i>ParameterType</i> was either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>DecimalDigits</i> was less than 0 or greater than the value for the argument <i>ParamDef</i> (precision).</p> <p>The value specified for <i>ParameterType</i> was SQL_C_TYPE_TIMESTAMP and the value for <i>ParameterType</i> was either SQL_CHAR or SQL_VARCHAR and the value for <i>DecimalDigits</i> was less than 0 or greater than 9.</p> <p>The value specified for <i>ParameterType</i> was SQL_C_TIMESTAMP_EXT and the value for <i>ParameterType</i> was either SQL_CHAR or SQL_VARCHAR and the value for <i>DecimalDigits</i> was less than 0 or greater than 12.</p>

## SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Table 11. SQLBindParameter SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY104	Invalid precision value.	The value specified for <i>ParameterType</i> was either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>ParamDef</i> was less than 1.
HY105	Invalid parameter type.	<i>InputOutputType</i> is not one of SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT.
HYC00	Driver not capable.	CLI or data source does not support the conversion specified by the combination of the value specified for the argument <i>ValueType</i> and the value specified for the argument <i>ParameterType</i> .  The value specified for the argument <i>ParameterType</i> is not supported by either CLI or the data source.

### Restrictions

SQLBindParameter() replaces the deprecated SQLSetParam() API in CLI V5 and above, and ODBC 2.0 and above.

An additional value for *StrLen\_or\_IndPtr*, SQL\_DEFAULT\_PARAM, was introduced in ODBC 2.0, to indicate that the procedure is to use the default value of a parameter, rather than a value sent from the application. Since DB2 stored procedure arguments do not support default values, specification of this value for *StrLen\_or\_IndPtr* argument will result in an error when the CALL statement is executed since the SQL\_DEFAULT\_PARAM value will be considered an invalid length.

ODBC 2.0 also introduced the SQL\_LEN\_DATA\_AT\_EXEC(*length*) macro to be used with the *StrLen\_or\_IndPtr* argument. The macro is used to specify the sum total length of the entire data that would be sent for character or binary C data via the subsequent SQLPutData() calls. Since the DB2 ODBC driver does not need this information, the macro is not needed. An ODBC application calls SQLGetInfo() with the SQL\_NEED\_LONG\_DATA\_LEN option to check if the driver needs this information. The DB2 ODBC driver will return 'N' to indicate that this information is not needed by SQLPutData().

### Example

```
SQLSMALLINT parameter1 = 0;

/* ... */

cliRC = SQLBindParameter(hstmt,
                        1,
                        SQL_PARAM_INPUT,
                        SQL_C_SHORT,
                        SQL_SMALLINT,
                        0,
                        0,
                        &parameter1,
                        0,
                        NULL);
```

## SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

### Purpose

Specification:	CLI 5.0	ODBC 1	
----------------	---------	--------	--

SQLBrowseConnect() supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source. Each call to SQLBrowseConnect() returns successive levels of attributes and attribute values. When all levels have been enumerated, a connection to the data source is completed and a complete connection string is returned by SQLBrowseConnect(). A return code of SQL\_SUCCESS or SQL\_SUCCESS\_WITH\_INFO indicates that all connection information has been specified and the application is now connected to the data source.

**Unicode Equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLBrowseConnectW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLBrowseConnect (
    SQLHDBC      ConnectionHandle,           /* hdbc */
    SQLCHAR      *InConnectionString,      /* *szConnStrIn */
    SQLSMALLINT  InConnectionStringLength,  /* dbConnStrIn */
    SQLCHAR      *OutConnectionString,     /* *szConnStrOut */
    SQLSMALLINT  OutConnectionStringCapacity, /* dbConnStrOutMax */
    SQLSMALLINT  *OutConnectionStringLengthPtr); /* *pcbConnStrOut */
```

### Function Arguments

Table 12. SQLBrowseConnect arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle.
SQLCHAR *	<i>InConnectionString</i>	input	Browse request connection string (see <i>InConnectionString</i> argument).
SQLSMALLINT	<i>InConnectionStringLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>InConnectionString</i> .
SQLCHAR *	<i>OutConnectionString</i>	output	Pointer to a buffer in which to return the browse result connection string (see <i>OutConnectionString</i> argument).
SQLSMALLINT	<i>OutConnectionStringCapacity</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>OutConnectionString</i> buffer.
SQLSMALLINT *	<i>OutConnectionStringLengthPtr</i>	output	The total number of elements (excluding the null termination character) available to return in <i>OutConnectionString</i> . If the number of elements available to return is greater than or equal to <i>OutConnectionStringCapacity</i> , the connection string in <i>OutConnectionString</i> is truncated to <i>OutConnectionStringCapacity</i> minus the length of a null termination character.



## SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

### Usage

SQLBrowseConnect() requires an allocated connection. If SQLBrowseConnect() returns SQL\_ERROR, outstanding connection information is discarded, and the connection is returned to an unconnected state.

When SQLBrowseConnect() is called for the first time on a connection, the browse request connection string must contain the DSN keyword.

On each call to SQLBrowseConnect(), the application specifies the connection attribute values in the browse request connection string. CLI returns successive levels of attributes and attribute values in the browse result connection string; it returns SQL\_NEED\_DATA as long as there are connection attributes that have not yet been enumerated in the browse request connection string. The application uses the contents of the browse result connection string to build the browse request connection string for the next call to SQLBrowseConnect(). All mandatory attributes (those not preceded by an asterisk in the *OutConnectionString* argument) must be included in the next call to SQLBrowseConnect(). Note that the application cannot simply copy the entire content of previous browse result connection strings when building the current browse request connection string; that is, it cannot specify different values for attributes set in previous levels.

When all levels of connection and their associated attributes have been enumerated, CLI returns SQL\_SUCCESS, the connection to the data source is complete, and a complete connection string is returned to the application. The connection string is suitable to use as an argument for SQLDriverConnect() in conjunction with the SQL\_DRIVER\_NOPROMPT option to establish another connection. The complete connection string cannot be used in another call to SQLBrowseConnect(), however; if SQLBrowseConnect() were called again, the entire sequence of calls would have to be repeated.

SQLBrowseConnect() also returns SQL\_NEED\_DATA if there are recoverable, nonfatal errors during the browse process, for example, an invalid password supplied by the application or an invalid attribute keyword supplied by the application. When SQL\_NEED\_DATA is returned and the browse result connection string is unchanged, an error has occurred and the application can call SQLGetDiagRec() to return the SQLSTATE for browse-time errors. This permits the application to correct the attribute and continue the browse.

An application can terminate the browse process at any time by calling SQLDisconnect(). CLI will terminate any outstanding connection information and return the connection to an unconnected state.

### InConnectionString argument

A browse request connection string has the following syntax:

connection-string ::= attribute[] | attribute: connection-string

attribute ::= attribute-keyword=attribute-value  
| DRIVER={attribute-value}

attribute-keyword ::= DSN | UID | PWD | NEWPWD  
| driver-defined-attribute-keyword

## SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

attribute-value ::= character-string  
driver-defined-attribute-keyword ::= identifier

where

- character-string has zero or more SQLCHAR or SQLWCHAR elements
- identifier has one or more SQLCHAR or SQLWCHAR elements
- attribute-keyword is case insensitive
- attribute-value might be case sensitive
- the value of the **DSN** keyword does not consist solely of blanks
- **NEWPWD** is used as part of a change password request. The application can either specify the new string to use, for example, **NEWPWD=anewpass**; or specify **NEWPWD=**; and rely on a dialog box generated by the DB2 CLI driver to prompt for the new password

Because of connection string and initialization file grammar, keywords and attribute values that contain the characters `[]{}(),;?*=!@` should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (`\`) character. For CLI Version 2, braces are required around the **DRIVER** keyword.

If any keywords are repeated in the browse request connection string, CLI uses the value associated with the first occurrence of the keyword. If the **DSN** and **DRIVER** keywords are included in the same browse request connection string, CLI uses whichever keyword appears first.

### OutConnectionString argument

The browse result connection string is a list of connection attributes. A connection attribute consists of an attribute keyword and a corresponding attribute value. The browse result connection string has the following syntax:

connection-string ::= attribute[;] | attribute; connection-string

attribute ::= [\*]attribute-keyword=attribute-value  
attribute-keyword ::= ODBC-attribute-keyword  
| driver-defined-attribute-keyword

ODBC-attribute-keyword = {UID | PWD}[[:localized-identifier]]  
driver-defined-attribute-keyword ::= identifier[:localized-identifier]

attribute-value ::= {attribute-value-list} | ?  
(The braces are literal; they are returned by CLI.)  
attribute-value-list ::= character-string [:localized-character string] | character-string [:localized-character string], attribute-value-list

where

- character-string and localized-character string have zero or more SQLCHAR or SQLWCHAR elements
- identifier and localized-identifier have one or more elements; attribute-keyword is case insensitive
- attribute-value might be case sensitive

Because of connection string and initialization file grammar, keywords, localized identifiers, and attribute values that contain the characters `[]{}(),;?*=!@` should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (`\`) character.

## SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

The browse result connection string syntax is used according to the following semantic rules:

- If an asterisk (\*) precedes an attribute-keyword, the attribute is optional, and can be omitted in the next call to `SQLBrowseConnect()`.
- The attribute keywords **UID** and **PWD** have the same meaning as defined in `SQLDriverConnect()`.
- When connecting to a DB2 database, only **DSN**, **UID** and **PWD** are required. Other keywords can be specified but do not affect the connection.
- ODBC-attribute-keywords and driver-defined-attribute-keywords include a localized or user-friendly version of the keyword. This might be used by applications as a label in a dialog box. However, **UID**, **PWD**, or the identifier alone must be used when passing a browse request string to CLI.
- The {attribute-value-list} is an enumeration of actual values valid for the corresponding attribute-keyword. Note that the braces ({} ) do not indicate a list of choices; they are returned by CLI. For example, it might be a list of server names or a list of database names.
- If the attribute-value is a single question mark (?), a single value corresponds to the attribute-keyword. For example, `UID=JohnS`; `PWD=Sesame`.
- Each call to `SQLBrowseConnect()` returns only the information required to satisfy the next level of the connection process. CLI associates state information with the connection handle so that the context can always be determined on each call.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_NEED_DATA`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 13. `SQLBrowseConnect` `SQLSTATE`s

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated.	The buffer <code>*OutConnectionString</code> was not large enough to return entire browse result connection string, so the string was truncated. The buffer <code>*OutConnectionStringLengthPtr</code> contains the length of the untruncated browse result connection string. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01S00	Invalid connection string attribute.	An invalid attribute keyword was specified in the browse request connection string ( <code>InConnectionString</code> ). (Function returns <code>SQL_NEED_DATA</code> .)  An attribute keyword was specified in the browse request connection string ( <code>InConnectionString</code> ) that does not apply to the current connection level. (Function returns <code>SQL_NEED_DATA</code> .)
01S02	Option value changed.	CLI did not support the specified value of the <code>ValuePtr</code> argument in <code>SQLSetConnectAttr()</code> and substituted a similar value. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08001	Unable to connect to data source.	CLI was unable to establish a connection with the data source.
08002	Connection in use.	The specified connection had already been used to establish a connection with a data source and the connection was open.

## SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

Table 13. *SQLBrowseConnect SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
08004	The application server rejected establishment of the connection.	The data source rejected the establishment of the connection for implementation defined reasons.
08S01	Communication link failure.	The communication link between CLI and the data source to which it was trying to connect failed before the function completed processing.
28000	Invalid authorization specification.	Either the user identifier or the authorization string or both as specified in the browse request connection string ( <i>InConnectionString</i> ) violated restrictions defined by the data source.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by <code>SQLGetDiagRec()</code> in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument <i>InConnectionStringLength</i> was less than 0 and was not equal to <code>SQL_NTS</code> .  The value specified for argument <i>OutConnectionStringCapacity</i> was less than 0.

### Restrictions

None.

### Example

```
SQLCHAR connInStr[255]; /* browse request connection string */
SQLCHAR outStr[1025]; /* browse result connection string*/

/* ... */

cliRC = SQL_NEED_DATA;
while (cliRC == SQL_NEED_DATA)
{
    /* get required attributes to connect to data source */
    cliRC = SQLBrowseConnect(hdbc,
                            connInStr,
                            SQL_NTS,
                            outStr,
                            sizeof(outStr),
                            &indicator);
    DBC_HANDLE_CHECK(hdbc, cliRC);

    printf(" So far, have connected %d times to database %s\n",
           count++, db1Alias);
    printf(" Resulting connection string:

    /* if inadequate connection information was provided, exit
       the program */
    if (cliRC == SQL_NEED_DATA)
    {
        printf(" You can provide other connection information "
```

## SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

```
        "here by setting connInStr\n");
    break;
}

/* if the connection was successful, output confirmation */
if (cliRC == SQL_SUCCESS)
{
    printf(" Connected to the database
}
}
```

## SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

### Purpose

Specification:	CLI 6.0	ODBC 3.0	
----------------	---------	----------	--

SQLBulkOperations() is used to perform the following operations on a keyset-driven cursor:

- Add new rows
- Update a set of rows where each row is identified by a bookmark
- Delete a set of rows where each row is identified by a bookmark
- Fetch a set of rows where each row is identified by a bookmark

### Syntax

```
SQLRETURN SQLBulkOperations (
    SQLHSTMT StatementHandle,
    SQLSMALLINT Operation);
```

### Function arguments

Table 14. SQLBulkOperations arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT	<i>Operation</i>	Input	Operation to perform: <ul style="list-style-type: none"><li>• SQL_ADD</li><li>• SQL_UPDATE_BY_BOOKMARK</li><li>• SQL_DELETE_BY_BOOKMARK</li><li>• SQL_FETCH_BY_BOOKMARK</li></ul>

### Usage

An application uses SQLBulkOperations() to perform the following operations on the base table or view that corresponds to the current query in a keyset-driven cursor:

- Add new rows
- Update a set of rows where each row is identified by a bookmark
- Delete a set of rows where each row is identified by a bookmark
- Fetch a set of rows where each row is identified by a bookmark

A generic application should first ensure that the required bulk operation is supported. To do so, it can call SQLGetInfo() with an *InfoType* of SQL\_DYNAMIC\_CURSOR\_ATTRIBUTES1 and SQL\_DYNAMIC\_CURSOR\_ATTRIBUTES2 (to see if SQL\_CA1\_BULK\_UPDATE\_BY\_BOOKMARK is returned, for instance).

## SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

After a call to `SQLBulkOperations()`, the block cursor position is undefined. The application has to call `SQLFetchScroll()` to set the cursor position. An application should only call `SQLFetchScroll()` with a *FetchOrientation* argument of `SQL_FETCH_FIRST`, `SQL_FETCH_LAST`, `SQL_FETCH_ABSOLUTE`, or `SQL_FETCH_BOOKMARK`. The cursor position is undefined if the application calls `SQLFetch()`, or `SQLFetchScroll()` with a *FetchOrientation* argument of `SQL_FETCH_PRIOR`, `SQL_FETCH_NEXT`, or `SQL_FETCH_RELATIVE`.

A column can be ignored in bulk operations (calls to `SQLBulkOperations()`). To do so, call `SQLBindCol()` and set the column length/indicator buffer (*StrLen\_or\_IndPtr*) to `SQL_COLUMN_IGNORE`. This does not apply to `SQL_DELETE_BY_BOOKMARK` bulk operation.

It is not necessary for the application to set the `SQL_ATTR_ROW_OPERATION_PTR` statement attribute when calling `SQLBulkOperations()` because rows cannot be ignored when performing bulk operations with this function.

The buffer pointed to by the `SQL_ATTR_ROWS_FETCHED_PTR` statement attribute contains the number of rows affected by a call to `SQLBulkOperations()`.

When the *Operation* argument is `SQL_ADD` or `SQL_UPDATE_BY_BOOKMARK`, and the select-list of the query specification associated with the cursor contains more than one reference to the same column, an error is generated.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_NEED_DATA`
- `SQL_STILL_EXECUTING`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 15. *SQLBulkOperations SQLSTATES*

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated.	The <i>Operation</i> argument was <code>SQL_FETCH_BY_BOOKMARK</code> , and string or binary data returned for a column or columns with a data type of <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> resulted in the truncation of non-blank character or non-NULL binary data.
01S07	Invalid conversion.	The <i>Operation</i> argument was <code>SQL_FETCH_BY_BOOKMARK</code> , the data type of the application buffer was not <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> , and the data returned to application buffers for one or more columns was truncated. (For numeric C data types, the fractional part of the number was truncated. For time and timestamp data types, the fractional portion of the time was truncated.)  (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)

## SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

Table 15. SQLBulkOperations SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07006	Restricted data type attribute violation.	<p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK, and the data value of a column in the result set could not be converted to the data type specified by the <i>TargetType</i> argument in the call to SQLBindCol().</p> <p>The <i>Operation</i> argument was SQL_UPDATE_BY_BOOKMARK or SQL_ADD, and the data value in the application buffers could not be converted to the data type of a column in the result set.</p>
07009	Invalid descriptor index.	The argument <i>Operation</i> was SQL_ADD and a column was bound with a column number greater than the number of columns in the result set, or the column number was less than 0.
21S02	Degree of derived table does not match column list.	The argument <i>Operation</i> was SQL_UPDATE_BY_BOOKMARK; and no columns were updatable because all columns were either unbound, read-only, or the value in the bound length/indicator buffer was SQL_COLUMN_IGNORE.
22001	String data right truncation.	The assignment of a character or binary value to a column in the result set resulted in the truncation of non-blank (for characters) or non-null (for binary) characters or bytes.
22003	Numeric value out of range.	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the assignment of a numeric value to a column in the result set caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>The argument <i>Operation</i> was SQL_FETCH_BY_BOOKMARK, and returning the numeric value for one or more bound columns would have caused a loss of significant digits.</p>
22007	Invalid datetime format.	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the assignment of a date or timestamp value to a column in the result set caused the year, month, or day field to be out of range.</p> <p>The argument <i>Operation</i> was SQL_FETCH_BY_BOOKMARK, and returning the date or timestamp value for one or more bound columns would have caused the year, month, or day field to be out of range.</p>
22008	Date/time field overflow.	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the performance of datetime arithmetic on data being sent to a column in the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar.</p> <p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK, and the performance of datetime arithmetic on data being retrieved from the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar.</p>

## SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

Table 15. *SQLBulkOperations SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
22018	Invalid character value for cast specification.	<p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK; the C type was an exact or approximate numeric or datetime data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.</p> <p>The argument <i>Operation</i> was SQL_ADD or SQL_UPDATE_BY_BOOKMARK; the SQL type was an exact or approximate numeric or datetime data type; the C type was SQL_C_CHAR; and the value in the column was not a valid literal of the bound SQL type.</p>
23000	Integrity constraint violation.	<p>The <i>Operation</i> argument was SQL_ADD, SQL_DELETE_BY_BOOKMARK, or SQL_UPDATE_BY_BOOKMARK, and an integrity constraint was violated.</p> <p>The <i>Operation</i> argument was SQL_ADD, and a column that was not bound is defined as NOT NULL and has no default.</p> <p>The <i>Operation</i> argument was SQL_ADD, the length specified in the bound <i>StrLen_or_IndPtr</i> buffer was SQL_COLUMN_IGNORE, and the column did not have a default value.</p>
24000	Invalid cursor state.	The <i>StatementHandle</i> was in an executed state but no result set was associated with the <i>StatementHandle</i> . SQLFetch() or SQLFetchScroll() was not called by the application after SQLExecute() or SQLExecDirect().
40001	Serialization failure.	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown.	The associated connection failed during the execution of this function and the state of the transaction cannot be determined.
42000	Syntax error or access violation.	CLI was unable to lock the row as needed to perform the operation requested in the <i>Operation</i> argument.
44000	WITH CHECK OPTION violation.	The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the insert or update was performed on a viewed table or a table derived from the viewed table which was created by specifying WITH CHECK OPTION, such that one or more rows affected by the insert or update will no longer be present in the viewed table.
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation error.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .



## SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

Table 15. *SQLBulkOperations SQLSTATES (continued)*

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called For the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY011	Operation invalid at this time.	The SQL_ATTR_ROW_STATUS_PTR statement attribute was set between calls to SQLFetch() or SQLFetchScroll() and SQLBulkOperations.
HY013	Unexpected memory handling error.	CLI was unable to access memory required to support execution or completion of this function.
HY090	Invalid string or buffer length.	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, a data value was not a null pointer; the C data type was SQL_C_BINARY or SQL_C_CHAR; and the column length value was less than 0, but not equal to SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The value in a length/indicator buffer was SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long data type; and the SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo() was "Y".</p> <p>The <i>Operation</i> argument was SQL_ADD, the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and column 0 was bound to a buffer whose length was not equal to the maximum length for the bookmark for this result set. (This length is available in the SQL_DESC_OCTET_LENGTH field of the IRD, and can be obtained by calling SQLDescribeCol(), SQLColAttribute(), or SQLGetDescField().)</p>

## SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

Table 15. SQLBulkOperations SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Invalid attribute identifier.	The value specified for the <i>Operation</i> argument was invalid.  The <i>Operation</i> argument was SQL_ADD, SQL_UPDATE_BY_BOOKMARK, or SQL_DELETE_BY_BOOKMARK, and the SQL_ATTR_CONCURRENCY statement attribute was set to SQL_CONCUR_READ_ONLY.  The <i>Operation</i> argument was SQL_DELETE_BY_BOOKMARK, SQL_FETCH_BY_BOOKMARK, or SQL_UPDATE_BY_BOOKMARK, and the bookmark column was not bound or the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
HYC00	Optional feature not implemented.	CLI or data source does not support the operation requested in the <i>Operation</i> argument.
HYT00	Timeout expired.	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr() with an <i>Attribute</i> argument of SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired.	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr(), SQL_ATTR_CONNECTION_TIMEOUT.

### Restrictions

None.

## SQLCancel function (CLI) - Cancel statement

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLCancel () can be used to prematurely terminate the data-at-execution sequence for sending and retrieving long data in pieces.

SQLCancel () can also be used to cancel a function called in a different thread.

### Syntax

```
SQLRETURN SQLCancel (SQLHSTMT StatementHandle); /* hstmt */
```

### Function arguments

Table 16. SQLCancel arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle

### Usage

After SQLExecDirect () or SQLExecute () returns SQL\_NEED\_DATA to solicit for values for data-at-execution parameters, SQLCancel () can be used to cancel the

## SQLCancel function (CLI) - Cancel statement

data-at-execution sequence for sending and retrieving long data in pieces. SQLCancel() can be called any time before the final SQLParamData() in the sequence. After the cancellation of this sequence, the application can call SQLExecute() or SQLExecDirect() to re-initiate the data-at-execution sequence.

If no processing is being done on the statement, SQLCancel() has no effect. Applications should not call SQLCancel() to close a cursor, but rather SQLFreeStmt() should be used.

### Canceling queries on host databases

To call SQLCancel() against a server which does not have native interrupt support (such as DB2 for z/OS<sup>®</sup>, Version 7 and earlier, and IBM DB2 for IBM i), the INTERRUPT\_ENABLED option must be set when cataloging the DCS database entry for the server.

When the INTERRUPT\_ENABLED option is set and SQLCancel() is received by the server, the server drops the connection and rolls back the unit of work. The application receives an SQL30081N error indicating that the connection to the server has been terminated. In order for the application to process additional database requests, the application must establish a new connection with the database server.

### Canceling asynchronous processing

After an application calls a function asynchronously, it calls the function repeatedly to determine whether it has finished processing. If the function is still processing, it returns SQL\_STILL\_EXECUTING.

After any call to the function that returns SQL\_STILL\_EXECUTING, an application can call SQLCancel() to cancel the function. If the cancel request is successful, SQL\_SUCCESS is returned. This message does not indicate that the function was actually canceled; it indicates that the cancel request was processed. The application must then continue to call the original function until the return code is not SQL\_STILL\_EXECUTING. If the function was successfully canceled, the return code is for that function is SQL\_ERROR and SQLSTATE HY008 (Operation was cancelled.). If the function succeeded by completing its normal processing, the return code is SQL\_SUCCESS or SQL\_SUCCESS\_WITH\_INFO. If the function failed for reasons other than cancellation, the return code is SQL\_ERROR and an SQLSTATE other than HY008 (Operation was cancelled.).

### Canceling functions in multithread applications

In a multithread application, the application can cancel a function that is running synchronously on a statement. To cancel the function, the application calls SQLCancel() with the same statement handle as that used by the target function, but on a different thread. How the function is canceled depends upon the operating system. The return code of the SQLCancel() call indicates only whether CLI processed the request successfully. Only SQL\_SUCCESS or SQL\_ERROR can be returned; no SQLSTATEs are returned. If the original function is canceled, it returns SQL\_ERROR and SQLSTATE HY008 (Operation was cancelled.).

If an SQL statement is being executed when SQLCancel() is called on another thread to cancel the statement execution, it is possible that the execution succeeds and returns SQL\_SUCCESS, while the cancel is also successful. In this case, CLI assumes that the cursor opened by the statement execution is closed by the cancel,

## SQLCancel function (CLI) - Cancel statement

so the application will not be able to use the cursor.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_INVALID\_HANDLE
- SQL\_ERROR

**Note:** SQL\_SUCCESS means that the cancel request was processed, not that the function call was canceled.

### Diagnostics

Table 17. SQLCancel SQLSTATEs

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY018	Server declined cancel request.	The server declined the cancel request.
HY506	Error closing a file.	An error occurred when closing the temporary file generated by CLI when inserting LOB data in pieces using SQLParamData()/SQLPutData().

### Restrictions

None.

### Example

```
/* cancel the SQL_DATA_AT_EXEC state for hstmt */  
cliRC = SQLCancel(hstmt);
```

## SQLCloseCursor function (CLI) - Close cursor and discard pending results

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLCloseCursor() closes a cursor that has been opened on a statement and discards pending results.

### Syntax

```
SQLRETURN SQLCloseCursor (SQLHSTMT StatementHandle); /* hStmt */
```

## SQLCloseCursor function (CLI) - Close cursor and discard pending results

### Function arguments

Table 18. SQLCloseCursor arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle

### Usage

After an application calls SQLCloseCursor(), the application can reopen the cursor later by executing a SELECT statement again with the same or different parameter values. SQLCloseCursor() can be called before a transaction is completed.

SQLCloseCursor() returns SQLSTATE 24000 (Invalid cursor state) if no cursor is open. Calling SQLCloseCursor() is equivalent to calling SQLFreeStmt() with the SQL\_CLOSE option, with the exception that SQLFreeStmt() with SQL\_CLOSE has no effect on the application if no cursor is open on the statement, while SQLCloseCursor() returns SQLSTATE 24000 (Invalid cursor state).

The statement attribute SQL\_ATTR\_CLOSE\_BEHAVIOR can be used to indicate whether or not CLI should attempt to release read locks acquired during a cursor's operation when the cursor is closed. If SQL\_ATTR\_CLOSE\_BEHAVIOR is set to SQL\_CC\_RELEASE then the database manager will attempt to release all read locks (if any) that have been held for the cursor.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 19. SQLCloseCursor SQLSTATEs

SQLSTATE	Description	Explanation
01000	General warning	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state.	No cursor was open on the <i>StatementHandle</i> . (This is returned only by CLI Version 5 or later.)
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called.  SQLExecute() or SQLExecDirect() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.

## SQLCloseCursor function (CLI) - Close cursor and discard pending results

Table 19. SQLCloseCursor SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

### Restrictions

None.

### Example

```
/* close the cursor */
cliRC = SQLCloseCursor(hstmt);
```

## SQLColAttribute function (CLI) - Return a column attribute

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLColAttribute() returns descriptor information for a column in a result set. Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLColAttributeW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

In a Windows 64-bit environment, the syntax is as follows:

```
SQLRETURN SQLColAttribute (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLSMALLINT       ColumnNumber,    /* icol */
    SQLSMALLINT       FieldIdentifier,  /* fDescType */
    SQLPOINTER        CharacterAttributePtr, /* rgbDesc */
    SQLSMALLINT       BufferLength,     /* cbDescMax */
    SQLSMALLINT       *StringLengthPtr, /* pcbDesc */
    SQLLEN            *NumericAttributePtr); /* pfDesc */
```

The syntax for all other platforms is as follows:

```
SQLRETURN SQLColAttribute (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLSMALLINT       ColumnNumber,    /* icol */
    SQLSMALLINT       FieldIdentifier,  /* fDescType */
    SQLPOINTER        CharacterAttributePtr, /* rgbDesc */
    SQLSMALLINT       BufferLength,     /* cbDescMax */
    SQLSMALLINT       *StringLengthPtr, /* pcbDesc */
    SQLPOINTER        NumericAttributePtr); /* pfDesc */
```

### Function arguments

Table 20. SQLColAttribute arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle.

## SQLColAttribute function (CLI) - Return a column attribute

Table 20. SQLColAttribute arguments (continued)

Data type	Argument	Use	Description
SQLUSMALLINT	<i>ColumnNumber</i>	input	The number of the record in the IRD from which the field value is to be retrieved. This argument corresponds to the column number of result data, ordered sequentially from left to right, starting at 1. Columns can be described in any order.  Column 0 can be specified in this argument, but all values except SQL_DESC_TYPE and SQL_DESC_OCTET_LENGTH will return undefined values.
SQLSMALLINT	<i>FieldIdentifier</i>	input	The field in row <i>ColumnNumber</i> of the IRD that is to be returned (see Table 21 on page 52).
SQLPOINTER	<i>CharacterAttributePtr</i>	output	Pointer to a buffer in which to return the value in the <i>FieldIdentifier</i> field of the <i>ColumnNumber</i> row of the IRD, if the field is a character string. Otherwise, the field is unused.
SQLINTEGER	<i>BufferLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>*CharacterAttributePtr</i> buffer, if the field is a character string. Otherwise, the field is ignored.
SQLSMALLINT *	<i>StringLengthPtr</i>	output	Pointer to a buffer in which to return the total number of bytes (excluding the byte count of the null termination character for character data) available to return in <i>*CharacterAttributePtr</i> .  For character data, if the number of bytes available to return is greater than or equal to <i>BufferLength</i> , the descriptor information in <i>*CharacterAttributePtr</i> is truncated to <i>BufferLength</i> minus the length of a null termination character and is null-terminated by CLI.  For all other types of data, the value of <i>BufferLength</i> is ignored and CLI assumes the size of <i>*CharacterAttributePtr</i> is 32 bits.
SQLLEN* (Window 64-bit) or SQLPOINTER	<i>NumericAttributePtr</i>	output	Pointer to a buffer in which to return the value in the <i>FieldIdentifier</i> field of the <i>ColumnNumber</i> row of the IRD, if the field is a numeric descriptor type, such as SQL_DESC_COLUMN_LENGTH. Otherwise, the field is unused.

### Usage

SQLColAttribute() returns information either in *\*NumericAttributePtr* or in *\*CharacterAttributePtr*. Integer information is returned in *\*NumericAttributePtr* as a 32-bit, signed value; all other formats of information are returned in *\*CharacterAttributePtr*. When information is returned in *\*NumericAttributePtr*, CLI ignores *CharacterAttributePtr*, *BufferLength*, and *StringLengthPtr*. When information is returned in *\*CharacterAttributePtr*, CLI ignores *NumericAttributePtr*.

SQLColAttribute() returns values from the descriptor fields of the IRD. The function is called with a statement handle rather than a descriptor handle. The values returned by SQLColAttribute() for the *FieldIdentifier* values listed below can also be retrieved by calling SQLGetDescField() with the appropriate IRD handle.

## SQLColAttribute function (CLI) - Return a column attribute

The currently defined descriptor types, the version of CLI in which they were introduced (perhaps with a different name), and the arguments in which information is returned for them are shown below; it is expected that more descriptor types will be defined to take advantage of different data sources.

CLI must return a value for each of the descriptor types. If a descriptor type does not apply to a data source, then, unless otherwise stated, CLI returns 0 in *\*StringLengthPtr* or an empty string in *\*CharacterAttributePtr*.

The following table lists the descriptor types returned by SQLColAttribute().

Table 21. SQLColAttribute arguments

<i>FieldIdentifier</i>	<b>Information returned in</b>	<b>Description</b>
SQL_DESC_AUTO_UNIQUE_VALUE (DB2 CLI/v2)	Numeric AttributePtr	Indicates if the column data type is an auto increment data type.  SQL_FALSE is returned in <i>NumericAttributePtr</i> for all DB2 SQL data types. Currently CLI is not able to determine if a column is an identity column, therefore SQL_FALSE is always returned. This limitation does not fully conform to the ODBC specifications. Future versions of CLI for UNIX and Windows servers will provide auto-unique support.
SQL_DESC_BASE_COLUMN_NAME (DB2 CLI/v5)	Character AttributePtr	The base column name for the set column. If a base column name does not exist (as in the case of columns that are expressions), then this variable contains an empty string.  This information is returned from the SQL_DESC_BASE_COLUMN_NAME record field of the IRD, which is a read-only field.
SQL_DESC_BASE_TABLE_NAME (DB2 CLI/v5)	Character AttributePtr	The name of the base table that contains the column. If the base table name cannot be defined or is not applicable, then this variable contains an empty string.
SQL_DESC_CASE_SENSITIVE (DB2 CLI/v2)	Numeric AttributePtr	Indicates if the column data type is a case sensitive data type.  Either SQL_TRUE or SQL_FALSE will be returned in <i>NumericAttributePtr</i> depending on the data type.  Case sensitivity does not apply to graphic data types, SQL_FALSE is returned.  SQL_FALSE is returned for non-character data types and for the XML data type.
SQL_DESC_CATALOG_NAME (DB2 CLI/v2)	Character AttributePtr	An empty string is returned since CLI only supports two part naming for a table.
SQL_DESC_CONCISE_TYPE (DB2 CLI/v5)	Numeric AttributePtr	The concise data type.  For the datetime data types, this field returns the concise data type, for example, SQL_TYPE_TIME.  This information is returned from the SQL_DESC_CONCISE_TYPE record field of the IRD.
SQL_DESC_COUNT (DB2 CLI/v2)	Numeric AttributePtr	The number of columns in the result set is returned in <i>NumericAttributePtr</i> .



## SQLColAttribute function (CLI) - Return a column attribute

Table 21. SQLColAttribute arguments (continued)

<i>FieldIdentifier</i>	<b>Information returned in</b>	<b>Description</b>
SQL_DESC_DISPLAY_SIZE (DB2 CLI/v2)	Numeric AttributePtr	The maximum number of bytes needed to display the data in character form is returned in <i>NumericAttributePtr</i> .  Refer to the data type display size table for the display size of each of the column types.
SQL_DESC_DISTINCT_TYPE (DB2 CLI/v2)	Character AttributePtr	The user defined distinct type name of the column is returned in <i>CharacterAttributePtr</i> . If the column is a built-in SQL type and not a user defined distinct type, an empty string is returned. <b>Note:</b> This is an IBM defined extension to the list of descriptor attributes defined by ODBC.
SQL_DESC_FIXED_PREC_SCALE (DB2 CLI/v2)	Numeric AttributePtr	SQL_TRUE if the column has a fixed precision and non-zero scale that are data-source-specific.  SQL_FALSE if the column does not have a fixed precision and non-zero scale that are data-source-specific.  SQL_FALSE is returned in <i>NumericAttributePtr</i> for all DB2 SQL data types.
SQL_DESC_LABEL (DB2 CLI/v2)	Character AttributePtr	The column label is returned in <i>CharacterAttributePtr</i> . If the column does not have a label, the column name or the column expression is returned. If the column is unlabeled and unnamed, an empty string is returned.
SQL_DESC_LENGTH (DB2 CLI/v2)	Numeric AttributePtr	A numeric value that is either the maximum or actual element (SQLCHAR or SQLWCHAR) length of a character string or binary data type. It is the maximum element length for a fixed-length data type, or the actual element length for a variable-length data type. Its value always excludes the null termination byte that ends the character string.  This information is returned from the SQL_DESC_LENGTH record field of the IRD.  This value is 0 for the XML data type.
SQL_DESC_LITERAL_PREFIX (DB2 CLI/v5)	Character AttributePtr	This VARCHAR(128) record field contains the character or characters that CLI recognizes as a prefix for a literal of this data type. This field contains an empty string for a data type for which a literal prefix is not applicable.
SQL_DESC_LITERAL_SUFFIX (DB2 CLI/v5)	Character AttributePtr	This VARCHAR(128) record field contains the character or characters that CLI recognizes as a suffix for a literal of this data type. This field contains an empty string for a data type for which a literal suffix is not applicable.

## SQLColAttribute function (CLI) - Return a column attribute

Table 21. SQLColAttribute arguments (continued)

FieldIdentifier	Information returned in	Description
SQL_DESC_LOCAL_TYPE_NAME (DB2 CLI/v5)	Character AttributePtr	This VARCHAR(128) record field contains any localized (native language) name for the data type that might be different from the regular name of the data type. If there is no localized name, then an empty string is returned. This field is for display purposes only. The character set of the string is locale-dependent and is typically the default character set of the server.
SQL_DESC_NAME (DB2 CLI/v2)	Character AttributePtr	<p>The name of the column <i>ColumnNumber</i> is returned in <i>CharacterAttributePtr</i>. If the column is an expression, then the column number is returned.</p> <p>In either case, SQL_DESC_UNNAMED is set to SQL_NAMED. If there is no column name or a column alias, an empty string is returned and SQL_DESC_UNNAMED is set to SQL_UNNAMED.</p> <p>This information is returned from the SQL_DESC_NAME record field of the IRD.</p>
SQL_DESC_NULLABLE (DB2 CLI/v2)	Numeric AttributePtr	<p>If the column identified by <i>ColumnNumber</i> can contain nulls, then SQL_NULLABLE is returned in <i>NumericAttributePtr</i>.</p> <p>If the column is constrained not to accept nulls, then SQL_NO_NULLS is returned in <i>NumericAttributePtr</i>.</p> <p>This information is returned from the SQL_DESC_NULLABLE record field of the IRD.</p>
SQL_DESC_NUM_PREX_RADIX (DB2 CLI/v5)	Numeric AttributePtr	<ul style="list-style-type: none"> <li>• If the data type in the SQL_DESC_TYPE field is an approximate data type, this SQLINTEGER field contains a value of 2 because the SQL_DESC_PRECISION field contains the number of bits.</li> <li>• If the data type in the SQL_DESC_TYPE field is an exact numeric data type, this field contains a value of 10 because the SQL_DESC_PRECISION field contains the number of decimal digits.</li> </ul>

## SQLColAttribute function (CLI) - Return a column attribute

Table 21. SQLColAttribute arguments (continued)

FieldIdentifier	Information returned in	Description
SQL_DESC_OCTET_LENGTH (DB2 CLI/v2)	Numeric AttributePtr	<p>The number of bytes of data associated with the column is returned in <i>NumericAttributePtr</i>. This is the length in bytes of data transferred on the fetch or SQLGetData() for this column if SQL_C_DEFAULT is specified as the C data type. Refer to data type length table for the length of each of the SQL data types.</p> <p>If the column identified in <i>ColumnNumber</i> is a fixed length character or binary string, (for example, SQL_CHAR or SQL_BINARY) the actual length is returned.</p> <p>If the column identified in <i>ColumnNumber</i> is a variable length character or binary string, (for example, SQL_VARCHAR or SQL_BLOB) the maximum length is returned.</p> <p>If the column identified in <i>ColumnNumber</i> is of type SQL_XML, 0 is returned.</p>
SQL_DESC_PRECISION (DB2 CLI/v2)	Numeric AttributePtr	<p>The precision in units of digits is returned in <i>NumericAttributePtr</i> if the column is SQL_DECIMAL, SQL_NUMERIC, SQL_DOUBLE, SQL_FLOAT, SQL_INTEGER, SQL_REAL or SQL_SMALLINT.</p> <p>If the column is a character SQL data type, then the precision returned in <i>NumericAttributePtr</i>, indicates the maximum number of SQLCHAR or SQLWCHAR elements the column can hold.</p> <p>If the column is a graphic SQL data type, then the precision returned in <i>NumericAttributePtr</i>, indicates the maximum number of double-byte elements the column can hold.</p> <p>If the column is the XML data type, the precision is 0.</p> <p>Refer to data type precision table for the precision of each of the SQL data types.</p> <p>This information is returned from the SQL_DESC_PRECISION record field of the IRD.</p>
SQL_DESC_SCALE (DB2 CLI/v2)	Numeric AttributePtr	<p>The scale attribute of the column is returned. Refer to the data type scale table for the scale of each of the SQL data types.</p> <p>This information is returned from the SCALE record field of the IRD.</p>
SQL_DESC_SCHEMA_NAME (DB2 CLI/v2)	Character AttributePtr	<p>The schema of the table that contains the column is returned in <i>CharacterAttributePtr</i>. The name of the schema that contains the table is returned. If the schema name is of less than 8 characters, then spaces are appended as extra characters.</p>

## SQLColAttribute function (CLI) - Return a column attribute

Table 21. SQLColAttribute arguments (continued)

FieldIdentifier	Information returned in	Description
SQL_DESC_SEARCHABLE (DB2 CLI/v2)	Numeric AttributePtr	Indicates if the column data type is searchable: <ul style="list-style-type: none"> <li>• SQL_PRED_NONE (SQL_UNSEARCHABLE in DB2 CLI/v2) if the column cannot be used in a WHERE clause.</li> <li>• SQL_PRED_CHAR (SQL_LIKE_ONLY in DB2 CLI/v2) if the column can be used in a WHERE clause only with the LIKE predicate.</li> <li>• SQL_PRED_BASIC (SQL_ALL_EXCEPT_LIKE in DB2 CLI/v2) if the column can be used in a WHERE clause with all comparison operators except LIKE.</li> <li>• SQL_SEARCHABLE if the column can be used in a WHERE clause with any comparison operator.</li> </ul>
SQL_DESC_TABLE_NAME (DB2 CLI/v2)	Character AttributePtr	The name of the table that contains the column is returned. If the table name cannot be defined or is not applicable, then this variable contains an empty string.
SQL_DESC_TYPE (DB2 CLI/v2)	Numeric AttributePtr	The SQL data type of the column identified in <i>ColumnNumber</i> is returned in <i>NumericAttributePtr</i> . The possible values returned are listed in table of symbolic and default data types for CLI.  When <i>ColumnNumber</i> is equal to 0, SQL_BINARY is returned for variable-length bookmarks, and SQL_INTEGER is returned for fixed-length bookmarks.  For the datetime data types, this field returns the verbose data type, for example, SQL_DATETIME.  This information is returned from the SQL_DESC_TYPE record field of the IRD.
SQL_DESC_TYPE_NAME (DB2 CLI/v2)	Character AttributePtr	The type of the column (as entered in an SQL statement) is returned in <i>CharacterAttributePtr</i> .  For information on each data type refer to the list of symbolic and default data types for CLI.
SQL_DESC_UNNAMED (DB2 CLI/v5)	Numeric AttributePtr	SQL_NAMED or SQL_UNNAMED. If the SQL_DESC_NAME field of the IRD contains a column alias, or a column name, SQL_NAMED is returned. If there is no column name or a column alias, SQL_UNNAMED is returned.  This information is returned from the SQL_DESC_UNNAMED record field of the IRD.
SQL_DESC_UNSIGNED (DB2 CLI/v2)	Numeric AttributePtr	Indicates if the column data type is an unsigned type or not.  SQL_TRUE is returned in <i>NumericAttributePtr</i> for all non-numeric data types, SQL_FALSE is returned for all numeric data types.

## SQLColAttribute function (CLI) - Return a column attribute

Table 21. SQLColAttribute arguments (continued)

FieldIdentifier	Information returned in	Description
SQL_DESC_UPDATABLE (DB2 CLI/v2)	Numeric AttributePtr	Indicates if the column data type is an updatable data type: <ul style="list-style-type: none"> <li>• SQL_ATTR_READONLY is returned if the result set column is read-only.</li> <li>• SQL_ATTR_WRITE is returned if the result set column is read-write.</li> <li>• SQL_ATTR_READWRITE_UNKNOWN is returned if it is not known whether the result set column is updatable or not.</li> </ul>

This function is an extensible alternative to SQLDescribeCol(). SQLDescribeCol() returns a fixed set of descriptor information based on ANSI-89 SQL. SQLColAttribute() allows access to the more extensive set of descriptor information available in ANSI SQL-92 and DBMS vendor extensions.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 22. SQLColAttribute SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The buffer <i>*CharacterAttributePtr</i> was not large enough to return the entire string value, so the string was truncated. The length of the untruncated string value is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
07005	The statement did not return a result set.	The statement associated with the <i>StatementHandle</i> did not return a result set. There were no columns to describe.
07009	Invalid descriptor index.	The value specified for <i>ColumnNumber</i> was equal to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF. The value specified for the argument <i>ColumnNumber</i> was greater than the number of columns in the result set.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

## SQLColAttribute function (CLI) - Return a column attribute

Table 22. SQLColAttribute SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The function was called prior to calling <code>SQLPrepare()</code> or <code>SQLExecDirect()</code> for the <i>StatementHandle</i> .  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.  <code>SQLExecute()</code> or <code>SQLExecDirect()</code> was called for the <i>StatementHandle</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
HY090	Invalid string or buffer length.	The value specified for the argument <i>BufferLength</i> was less than 0.
HY091	Invalid descriptor field identifier.	The value specified for the argument <i>FieldIdentifier</i> was not one of the defined values, and was not an implementation-defined value.
HYC00	Driver not capable.	The value specified for the argument <i>FieldIdentifier</i> was not supported by CLI.

`SQLColAttribute()` can return any SQLSTATE that can be returned by `SQLPrepare()` or `SQLExecute()` when called after `SQLPrepare()` and before `SQLExecute()` depending on when the data source evaluates the SQL statement associated with the *StatementHandle*.

For performance reasons, an application should not call `SQLColAttribute()` before executing a statement.

### Restrictions

None.

### Example

```
/* get display size for column */
cliRC = SQLColAttribute(hstmt,
                       (SQLSMALLINT)(i + 1),
                       SQL_DESC_DISPLAY_SIZE,
                       NULL,
                       0,
                       NULL,
                       &colDataDisplaySize)
```

---

## SQLColAttributes function (CLI) - Get column attributes

### Deprecated

#### Note:

In ODBC 3.0, `SQLColAttributes()` has been deprecated and replaced with `SQLColAttribute()`.

## SQLColAttributes function (CLI) - Get column attributes

Although this version of CLI continues to support `SQLColAttributes()`, use `SQLColAttribute()` in your CLI programs so that they conform to the latest standards.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLColAttributesW()`. Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Migrating to the new function

The statement:

```
SQLColAttributes (hstmt, colNum, SQL_DESC_COUNT, NULL, len,  
                NULL, &numCols);
```

for example, would be rewritten using the new function as:

```
SQLColAttribute (hstmt, colNum, SQL_DESC_COUNT, NULL, len,  
                NULL, &numCols);
```

---

## SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table

`SQLColumnPrivileges()` returns a list of columns and associated privileges for the specified table. The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to process a result set that is generated from a query.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLColumnPrivilegesW()`. See “Unicode functions (CLI)” on page 5 for information about ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLColumnPrivileges(  
    SQLHSTMT          StatementHandle, /* hstmt */  
    SQLCHAR           *CatalogName,   /* szCatalogName */  
    SQLSMALLINT       NameLength1,    /* cbCatalogName */  
    SQLCHAR           *SchemaName,    /* szSchemaName */  
    SQLSMALLINT       NameLength2,    /* cbSchemaName */  
    SQLCHAR           *TableName,     /* szTableName */  
    SQLSMALLINT       NameLength3,    /* cbTableName */  
    SQLCHAR           *ColumnName,    /* szColumnName */  
    SQLSMALLINT       NameLength4);  /* cbColumnName */
```

### Function arguments

Table 23. *SQLColumnPrivileges* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.

## SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table

Table 23. SQLColumnPrivileges arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	The catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS is returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or SQL_NTS if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	The schema qualifier of the table name.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	The table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableName</i> , or SQL_NTS if <i>TableName</i> is null-terminated.
SQLCHAR *	<i>ColumnName</i>	Input	A buffer that might contain a <i>pattern value</i> to qualify the result set by column name.
SQLSMALLINT	<i>NameLength4</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ColumnName</i> , or SQL_NTS if <i>ColumnName</i> is null-terminated.

### Usage

The results are returned as a standard result set that contains the columns listed in Columns Returned by SQLColumnPrivileges. The result set is ordered by TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, COLUMN\_NAME, and PRIVILEGE. If multiple privileges are associated with any given column, each privilege is returned as a separate row. A typical application might want to call this function after a call to SQLColumns() to determine column privilege information. The application should use the character strings that are returned in the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, COLUMN\_NAME columns of the SQLColumns() result set as input arguments to this function.

Because calls to SQLColumnPrivileges(), in many cases, map to a complex and thus expensive query against the system catalog, you should use the calls sparingly, and save the results rather than repeating the calls.

The *ColumnName* input argument accepts a search pattern, however, all other input arguments do not.

Sometimes, an application calls the function and no attempt is made to restrict the result set that is returned. In order to help reduce the long retrieval times, you can specify the configuration keyword SchemaList in the CLI initialization file to help restrict the result set when the application has supplied a null pointer for *SchemaName*. If the application specifies a *SchemaName* string, the SchemaList



## SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table

keyword is still used to restrict the output. Therefore, if the schema name that is supplied is not in the SchemaList string, the result is an empty result set.

In Version 9.7 Fix Pack 5, you can specify \*ALL or \*USRLIBL as values in the *SchemaName* to resolve unqualified stored procedure calls or to find libraries in catalog API calls. If you specify \*ALL, CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. For IBM DB2 for IBM i servers, if you specify \*USRLIBL, CLI searches on the current libraries of the server job. For other DB2 servers, \*USRLIBL does not have a special meaning and CLI searches using \*USRLIBL as a pattern. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL or \*USRLIBL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns returned by SQLColumnPrivileges

**Column 1 TABLE\_CAT (VARCHAR(128) Data type)**

Name of the catalog. The value is NULL if this table does not have catalogs.

**Column 2 TABLE\_SCHEM (VARCHAR(128))**

Name of the schema containing TABLE\_NAME.

**Column 3 TABLE\_NAME (VARCHAR(128) not NULL)**

Name of the table or view.

**Column 4 COLUMN\_NAME (VARCHAR(128) not NULL)**

Name of the column of the specified table or view.

**Column 5 GRANTOR (VARCHAR(128))**

Authorization ID of the user who granted the privilege.

**Column 6 GRANTEE (VARCHAR(128))**

Authorization ID of the user to whom the privilege is granted.

**Column 7 PRIVILEGE (VARCHAR(128))**

The column privilege. This can be:

- INSERT
- REFERENCES
- SELECT
- UPDATE

**Note:** Some IBM RDBMSs do not offer column level privileges at the column level. DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, and DB2 Server for VM and VSE support the UPDATE column privilege; there is one row in this result set for each updateable column. For all other privileges for DB2 Database for Linux, UNIX, and Windows, DB2 for z/OS, and DB2 Server for VM and VSE, and for all privileges for other IBM RDBMSs, if a privilege has been granted at the table level, a row is present in this result set.

**Column 8 IS\_GRANTABLE (VARCHAR(3) Data type)**

Indicates whether the grantee is permitted to grant the privilege to other users.

Either "YES" or "NO".

## SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table

**Note:** The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the SQLColumnPrivileges() result set in ODBC.

If there is more than one privilege associated with a column, each privilege is returned as a separate row in the result set.

### Return Codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 24. SQLColumnPrivileges SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	<i>TableName</i> is NULL.
HY010	Function sequence error	An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.  SQLExecute(), SQLExecDirect(), or SQLSetPos() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

## SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table

### Example

```
cliRC = SQLColumnPrivileges(hstmt,
                             NULL,
                             0,
                             tbSchema,
                             SQL_NTS,
                             tbName,
                             SQL_NTS,
                             colNamePattern,
                             SQL_NTS);
```

---

## SQLColumns function (CLI) - Get column information for a table

The `SQLColumns()` function returns a list of columns in the specified tables. The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to fetch a result set that is generated by a query.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

**Unicode Equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLColumnsW()`. For details about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLColumns (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *CatalogName, /* szCatalogName */
    SQLSMALLINT NameLength1, /* cbCatalogName */
    SQLCHAR *SchemaName, /* szSchemaName */
    SQLSMALLINT NameLength2, /* cbSchemaName */
    SQLCHAR *TableName, /* szTableName */
    SQLSMALLINT NameLength3, /* cbTableName */
    SQLCHAR *ColumnName, /* szColumnName */
    SQLSMALLINT NameLength4); /* cbColumnName */
```

### Function arguments

Table 25. *SQLColumns* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and <code>SQL_SUCCESS</code> will be returned. Otherwise, this is a valid filter for DBMSs that supports 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that is required to store <i>CatalogName</i> , or <code>SQL_NTS</code> if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that might contain a <i>pattern value</i> to qualify the result set by schema name.

## SQLColumns function (CLI) - Get column information for a table

Table 25. SQLColumns arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	A buffer that might contain a <i>pattern value</i> to qualify the result set by table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableName</i> , or SQL_NTS if <i>TableName</i> is null-terminated.
SQLCHAR *	<i>ColumnName</i>	Input	A buffer that might contain a <i>pattern value</i> to qualify the result set by column name.
SQLSMALLINT	<i>NameLength4</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ColumnName</i> , or SQL_NTS if <i>ColumnName</i> is null-terminated.

### Usage

Use this function to retrieve information about the columns of either a table or a set of tables. An application can call this function after a call to `SQLTables()` to determine the columns of a table. The application must use the character strings that are returned in the `TABLE_SCHEMA` and `TABLE_NAME` columns of the `SQLTables()` result set as input to this function.

The `SQLColumns()` function returns a standard result set that is ordered by `TABLE_CAT`, `TABLE_SCHEM`, `TABLE_NAME`, and `ORDINAL_POSITION`. Columns returned by `SQLColumns` lists the columns that are in the result set.

The *SchemaName*, *TableName*, and *ColumnName* input arguments accept search patterns.

Sometimes, an application calls the function and no attempt is made to restrict the result set that is returned. For some data sources that contain a large quantity of tables, views, and aliases for example, this scenario maps to an extremely large result set and very long retrieval times. In order to help reduce the long retrieval times, you can specify the configuration keyword **SchemaList** in the CLI initialization file to help restrict the result set when the application has supplied a null pointer for the **SchemaName**. If the application specifies a **SchemaName** string, the **SchemaList** keyword is still used to restrict the output. Therefore, if the schema name supplied is not in the **SchemaList** string, the result will be an empty result set.

This function does not return information about the columns of a result set. Instead, you should use `SQLDescribeCol()` or `SQLColAttribute()` function.

If the `SQL_ATTR_LONGDATA_COMPAT` attribute is set to `SQL_LD_COMPAT_YES` via either a call to `SQLSetConnectAttr()` or by setting the `LONGDATACOMPAT` keyword in the CLI initialization file, then the LOB data types are reported as `SQL_LONGVARCHAR`, `SQL_LONGVARBINARY` or `SQL_LONGVARGRAPHIC`.

## SQLColumns function (CLI) - Get column information for a table

In many cases, calls to the SQLColumns() function map to a complex and thus expensive query against the system catalog, so you should use the calls sparingly, and save the results rather than repeating calls.

Call SQLGetInfo() with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_OWNER\_SCHEMA\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

In Version 9.7 Fix Pack 5, you can specify \*ALL or \*USRLIBL as values in the *SchemaName* to resolve unqualified stored procedure calls or to find libraries in catalog API calls. If you specify \*ALL, CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. For IBM DB2 for IBM i servers, if you specify \*USRLIBL, CLI searches on the current libraries of the server job. For other DB2 servers, \*USRLIBL does not have a special meaning and CLI searches using \*USRLIBL as a pattern. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL or \*USRLIBL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns returned by SQLColumns

#### Column 1 TABLE\_CAT (VARCHAR(128))

The name of the catalog. The value is NULL if this table does not have catalogs.

#### Column 2 TABLE\_SCHEM (VARCHAR(128))

The name of the schema containing TABLE\_NAME.

#### Column 3 TABLE\_NAME (VARCHAR(128) not NULL)

The name of the table, view, alias, or synonym.

#### Column 4 COLUMN\_NAME (VARCHAR(128) not NULL)

The column identifier. The name of the column of the specified table, view, alias, or synonym.

#### Column 5 DATA\_TYPE (SMALLINT not NULL)

The SQL data type of the column that is identified by COLUMN\_NAME. The DATA\_TYPE is one of the values in the Symbolic SQL Data Type column in the table of symbolic and default data types for CLI.

#### Column 6 TYPE\_NAME (VARCHAR(128) not NULL)

A character string that represents the name of the data type that corresponds to DATA\_TYPE.

#### Column 7 COLUMN\_SIZE (INTEGER)

If the DATA\_TYPE column value denotes a character or binary string, this column contains the maximum length in SQLCHAR or SQLWCHAR elements for the column.

For date, time, and timestamp data types, the COLUMN\_SIZE is the total number of SQLCHAR or SQLWCHAR elements that are required to display the value when converted to character data type.

## SQLColumns function (CLI) - Get column information for a table

For numeric data types, the COLUMN\_SIZE is either the total number of digits or the total number of bits that are allowed in the column, depending on the value in the NUM\_PREC\_RADIX column in the result set.

For the XML data type, the length of zero is returned.

See the table of data type precision.

### Column 8 BUFFER\_LENGTH (INTEGER)

The maximum number of bytes for the associated C buffer to store data from this column if SQL\_C\_DEFAULT is specified on the SQLBindCol(), SQLGetData() and SQLBindParameter() calls. This length does not include any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

See the table of data type lengths.

### Column 9 DECIMAL\_DIGITS (SMALLINT)

The scale of the column. NULL is returned for data types where scale is not applicable.

See the table of data type scale.

### Column 10 NUM\_PREC\_RADIX (SMALLINT)

Either 10, 2, or NULL. If DATA\_TYPE is an approximate numeric data type, this column contains the value 2, and the COLUMN\_SIZE column contains the number of bits that are allowed in the column.

If DATA\_TYPE is an exact numeric data type, this column contains the value 10, and the COLUMN\_SIZE contains the number of decimal digits that are allowed for the column.

For numeric data types, the DBMS can return a NUM\_PREC\_RADIX of 10 or 2.

NULL is returned for data types where the radix is not applicable.

### Column 11 NULLABLE (SMALLINT not NULL)

SQL\_NO\_NULLS if the column does not accept NULL values.

SQL\_NULLABLE if the column accepts NULL values.

### Column 12 REMARKS (VARCHAR(254))

Might contain descriptive information about the column. It is possible that no information is returned in this column. For more details, see Optimize SQL columns keyword and attribute.

### Column 13 COLUMN\_DEF (VARCHAR(254))

The default value of the column. If the default value is a numeric literal, this column contains the character representation of the numeric literal with no enclosing single quotation marks. If the default value is a character string, this column is that string that is enclosed in single quotation marks. If the default value is a *pseudo-literal*, such as for DATE, TIME, and TIMESTAMP columns, this column contains the keyword of the pseudo-literal (for example, CURRENT DATE) with no enclosing quotation marks.

If NULL is specified as the default value, this column returns the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, this column contains TRUNCATED with no enclosing single quotation marks. If no default value is specified, this column is NULL.

## SQLColumns function (CLI) - Get column information for a table

It is possible that no information is returned in this column. For more details, see Optimize SQL columns keyword and attribute.

### Column 14 SQL\_DATA\_TYPE (SMALLINT not NULL)

The SQL data type, as it is displayed in the SQL\_DESC\_TYPE record field in the IRD. This column is the same as the DATA\_TYPE column in Columns returned by SQLColumns for the date, time, and timestamp data types.

### Column 15 SQL\_DATETIME\_SUB (SMALLINT)

The subtype code for datetime data types:

- SQL\_CODE\_DATE
- SQL\_CODE\_TIME
- SQL\_CODE\_TIMESTAMP

For all other data types this column returns NULL.

### Column 16 CHAR\_OCTET\_LENGTH (INTEGER)

For single byte character sets, this is the same as COLUMN\_SIZE. For the XML type, zero is returned. For all other data types, NULL is returned.

### Column 17 ORDINAL\_POSITION (INTEGER not NULL)

The ordinal position of the column in the table. The first column in the table is number 1.

### Column 18 IS\_NULLABLE (VARCHAR(254))

Contains the string 'NO' if the column is known to be not nullable, and 'YES' if the column is nullable.

**Note:** This result set is identical to the X/Open CLI Columns() result set specification, which is an extended version of the SQLColumns() result set that is specified in ODBC V2. The ODBC SQLColumns() result set includes every column in the same position.

## Optimize SQL columns keyword and attribute

It is possible to set up the CLI/ODBC Driver to optimize calls to the SQLColumns() function by using either:

- OPTIMIZE\_SQL\_COLUMNS CLI/ODBC configuration keyword
- SQL\_ATTR\_OPTIMIZE\_SQL\_COLUMNS connection attribute of SQLSetConnectAttr()

If either of these values are set, the information that is contained in the succeeding columns is not returned:

- Column 12 REMARKS
- Column 13 COLUMN\_DEF

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

## SQLColumns function (CLI) - Get column information for a table

### Diagnostics

Table 26. SQLColumns SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (<code>SQLParamData()</code>, <code>SQLPutData()</code>) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name-length arguments was less than 0, but not equal to <code>SQL_NTS</code> .
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

### Restriction

The `SQLColumns()` function does not support returning data from an alias of an alias. When called against an alias of an alias, the `SQLColumns()` function returns an empty result set.

### Example

```
/* get column information for a table */
cliRC = SQLColumns(hstmt,
                  NULL,
                  0,
                  tbSchemaPattern,
                  SQL_NTS,
                  tbNamePattern,
                  SQL_NTS,
                  colNamePattern,
                  SQL_NTS);
```



## SQLConnect function (CLI) - Connect to a data source

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLConnect() establishes a connection or a trusted connection to the target database. The application must supply a target SQL database, and optionally an authorization-name and an authentication-string.

A connection must be established before allocating a statement handle using SQLAllocHandle().

**Unicode Equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLConnectW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLConnect (
    SQLHDBC          ConnectionHandle, /* hdbc */
    SQLCHAR          *ServerName,      /* szDSN */
    SQLSMALLINT      ServerNameLength, /* cbDSN */
    SQLCHAR          *UserName,        /* szUID */
    SQLSMALLINT      UserNameLength,   /* cbUID */
    SQLCHAR          *Authentication,  /* szAuthStr */
    SQLSMALLINT      AuthenticationLength); /* cbAuthStr */
```

### Function arguments

Table 27. SQLConnect arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle
SQLCHAR *	<i>ServerName</i>	input	Data Source: The name or alias-name of the database.
SQLSMALLINT	<i>ServerNameLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>ServerName</i> argument.
SQLCHAR *	<i>UserName</i>	input	Authorization-name (user identifier)
SQLSMALLINT	<i>UserNameLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>UserName</i> argument.
SQLCHAR *	<i>Authentication</i>	input	Authentication-string (password)
SQLSMALLINT	<i>AuthenticationLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>Authentication</i> argument.

### Usage

The target database (also known as *data source*) for IBM RDBMSs is the database-alias. The application can obtain a list of databases available to connect to by calling SQLDataSources().

## SQLConnect function (CLI) - Connect to a data source

The input length arguments to `SQLConnect()` (*ServerNameLength*, *UserNameLength*, *AuthenticationLength*) can be set to the actual length of their associated data in elements (SQLCHAR or SQLWCHAR), not including any null-terminating character, or to SQL\_NTS to indicate that the associated data is null-terminated.

The *ServerName* and *UserName* argument values must not contain any blanks.

Stored procedures written using CLI must make a *null* `SQLConnect()` call. A null `SQLConnect()` is where the *ServerName*, *UserName*, and *Authentication* argument pointers are all set to NULL and their respective length arguments all set to 0. A null `SQLConnect()` still requires `SQLAllocHandle()` to be called first, but does not require that `SQLEndTran()` be called before `SQLDisconnect()`.

To create a trusted connection, specify the connection attribute `SQL_ATTR_USE_TRUSTED_CONTEXT` before calling `SQLConnect()`. If the database server accepts the connection as trusted the connection is treated as a trusted connection. Otherwise the connection is a regular connection and a warning is returned.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 28. SQLConnect SQLSTATES

SQLSTATE	Description	Explanation
01679	Unable to establish a trusted connection.	CLI requested a trusted connection but the trust attributes of the connection do not match any trusted context object on the database server. The connection is allowed but it is a regular connection, not a trusted connection.
08001	Unable to connect to data source.	CLI was unable to establish a connection with the data source (server).  The connection request was rejected because an existing connection established via embedded SQL already exists.
08002	Connection in use.	The specified <i>ConnectionHandle</i> has already been used to establish a connection with a data source and the connection is still open.
08004	The application server rejected establishment of the connection.	The data source (server) rejected the establishment of the connection.
28000	Invalid authorization specification.	The value specified for the argument <i>UserName</i> or the value specified for the argument <i>Authentication</i> violated restrictions defined by the data source.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

## SQLConnect function (CLI) - Connect to a data source

Table 28. SQLConnect SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	<p>The value specified for argument <i>ServerNameLength</i> was less than 0, but not equal to SQL_NTS and the argument <i>ServerName</i> was not a null pointer.</p> <p>The value specified for argument <i>UserNameLength</i> was less than 0, but not equal to SQL_NTS and the argument <i>UserName</i> was not a null pointer.</p> <p>The value specified for argument <i>AuthenticationLength</i> was less than 0, but not equal to SQL_NTS and the argument <i>Authentication</i> was not a null pointer.</p>
HY501	Invalid data source name.	An invalid data source name was specified in argument <i>ServerName</i> .
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

The implicit connection (or default database) option for IBM RDBMSs is not supported. SQLConnect() must be called before any SQL statements can be executed.

### Example

```
/* connect to the database */
cliRC = SQLConnect(hdbc,
                  (SQLCHAR *)db1Alias,
                  SQL_NTS,
                  (SQLCHAR *)user,
                  SQL_NTS,
                  (SQLCHAR *)pswd,
                  SQL_NTS);
```

## SQLCopyDesc function (CLI) - Copy descriptor information between handles

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLCopyDesc() copies descriptor information from one descriptor handle to another.

### Syntax

```
SQLRETURN SQLCopyDesc (
    SQLHDESC SourceDescHandle, /* hDescSource */
    SQLHDESC TargetDescHandle); /* hDescTarget */
```

## SQLCopyDesc function (CLI) - Copy descriptor information between handles

### Function arguments

Table 29. SQLCopyDesc arguments

Data type	Argument	Use	Description
SQLHDESC	<i>SourceDescHandle</i>	input	Source descriptor handle.
SQLHDESC	<i>TargetDescHandle</i>	input	Target descriptor handle. <i>TargetDescHandle</i> can be a handle to an application descriptor or an IPD. SQLCopyDesc() will return SQLSTATE HY016 (Cannot modify an implementation descriptor) if <i>TargetDescHandle</i> is a handle to an IRD.

### Usage

A call to SQLCopyDesc() copies the fields of the source descriptor handle to the target descriptor handle. Fields can only be copied to an application descriptor or an IPD, but not to an IRD. Fields can be copied from either an application or an implementation descriptor.

All fields of the descriptor, except SQL\_DESC\_ALLOC\_TYPE (which specifies whether the descriptor handle was automatically or explicitly allocated), are copied, whether or not the field is defined for the destination descriptor. Copied fields overwrite the existing fields in the *TargetDescHandle*.

All descriptor fields are copied, even if *SourceDescHandle* and *TargetDescHandle* are on two different connections or environments.

The call to SQLCopyDesc() is immediately aborted if an error occurs.

When the SQL\_DESC\_DATA\_PTR field is copied, a consistency check is performed. If the consistency check fails, SQLSTATE HY021 (Inconsistent descriptor information.) is returned and the call to SQLCopyDesc() is immediately aborted.

**Note:** Descriptor handles can be copied across connections or environments. An application may, however, be able to associate an explicitly allocated descriptor handle with a *StatementHandle*, rather than calling SQLCopyDesc() to copy fields from one descriptor to another. An explicitly allocated descriptor can be associated with another *StatementHandle* on the same *ConnectionHandle* by setting the SQL\_ATTR\_APP\_ROW\_DESC or SQL\_ATTR\_APP\_PARAM\_DESC statement attribute to the handle of the explicitly allocated descriptor. When this is done, SQLCopyDesc() does not have to be called to copy descriptor field values from one descriptor to another.

A descriptor handle cannot be associated with a *StatementHandle* on another *ConnectionHandle*, however; to use the same descriptor field values on *StatementHandle* on different *ConnectionHandle*, SQLCopyDesc() has to be called.

### Copying rows between tables

An ARD on one statement handle can serve as the APD on another statement handle. This allows an application to copy rows between tables without copying data at the application level. To do this, an application calls SQLCopyDesc() to copy the fields of an ARD that describes a fetched row of a table, to the APD for a parameter in an INSERT statement on another statement handle. The SQL\_ACTIVE\_STATEMENTS *InfoType* returned by the driver for a call to

## SQLCopyDesc function (CLI) - Copy descriptor information between handles

SQLGetInfo() must be greater than 1 for this operation to succeed.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

When SQLCopyDesc() returns SQL\_ERROR or SQL\_SUCCESS\_WITH\_INFO, an associated SQLSTATE value may be obtained by calling SQLGetDiagRec() with a HandleType of SQL\_HANDLE\_DESC and a Handle of *TargetDescHandle*. If an invalid *SourceDescHandle* was passed in the call, SQL\_INVALID\_HANDLE will be returned, but no SQLSTATE will be returned.

When an error is returned, the call to SQLCopyDesc() is immediately aborted, and the contents of the fields in the *TargetDescHandle* descriptor are undefined.

Table 30. SQLCopyDesc SQLSTATES

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The communication link between CLI and the data source to which it was trying to connect failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY007	Associated statement is not prepared.	<i>SourceDescHandle</i> was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.
HY016	Cannot modify an implementation row descriptor.	<i>TargetDescHandle</i> was associated with an IRD.
HY021	Inconsistent descriptor information.	The descriptor information checked during a consistency check was not consistent.
HY092	Option type out of range.	The call to SQLCopyDesc() prompted a call to SQLSetDescField(), but *ValuePtr was not valid for the <i>FieldIdentifier</i> argument on <i>TargetDescHandle</i> .

## SQLCopyDesc function (CLI) - Copy descriptor information between handles

### Restrictions

None.

### Example

```
SQLHANDLE hIRD, hARD; /* descriptor handles */

/* ... */

/* copy descriptor information between handles */
rc = SQLCopyDesc(hIRD, hARD);
```

---

## SQLCreateDb function (CLI) - Create a database

### Purpose

Specification:	CLI V9.7	ODBC	ISO CLI
----------------	----------	------	---------

The SQLCreateDb() function creates a database by using the specified database name, code set, and mode.

An active connection to the server must exist before you issue the SQLCreateDb API.

**Unicode equivalent:** The corresponding Unicode function is the SQLCreateDbW() function. For information about ANSI to Unicode function mappings, refer to “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQL_API_FN SQLCreateDb ( SQLHDBC           hDbc,
                                   SQLCHAR           *szDbName,
                                   SQLINTEGER        cbDbName,
                                   SQLCHAR           *szCodeSet,
                                   SQLINTEGER        cbCodeSet,
                                   SQLCHAR           *szMode,
                                   SQLINTEGER        cbMode);
```

### Function arguments

Table 31. SQLCreateDb arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hDbc</i>	input	Connection handle.
SQLCHAR *	<i>szDbName</i>	input	Name of the database that is to be created.
SQLINTEGER	<i>cbDbName</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of the function) that is needed to store the <i>szDbName</i> argument or to store SQL_NTS if the <i>szDbName</i> argument is null terminated.
SQLCHAR *	<i>szCodeSet</i>	input	Database code set information. <b>Note:</b> If the value of the <i>szCodeSet</i> argument is NULL, the database is created in the Unicode code page for DB2 data servers and in the UTF-8 code page for IDS data servers.

## SQLCreateDb function (CLI) - Create a database

Table 31. SQLCreateDb arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>cbCodeSet</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of the function) that is needed to store the <i>szCodeSet</i> argument or to store SQL_NTS if <i>szCodeSet</i> argument is null terminated.
SQLCHAR *	<i>szMode</i>	input	Database logging mode. <b>Note:</b> This value is applicable only to IDS data servers.
SQLINTEGER	<i>cbMode</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of the function) that is needed to store the <i>szMode</i> argument or to store SQL_NTS if <i>szMode</i> argument is null terminated.

### Usage

When creating a DB2 database, CLI application must first connect to the server instance by specifying the ATTACH keyword. The valid APIs, after connecting to the server instance using ATTACH keyword are SQLCreateDb(), SQLDropDb(), and SQLDisconnect(). Before performing other CLI operations on the new database, you must disconnect from the server instance and then connect to the new database.

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR

### Diagnostics

Table 32. SQLCreateDb SQLSTATEs

SQLSTATE	Description	Explanation
08003	Connection is closed.	The connection that was specified for the <i>SQLCreateDb</i> argument was not open.
HY090	Invalid string or buffer length.	The <i>cbDbName</i> , <i>cbCodeSet</i> , and <i>cbMode</i> arguments have a maximum length of 128. If you specify an invalid value, CLI generates an error.

### Restrictions

- A connection representing an instance attachment is required.
- The SQLCreateDb() function is not supported for DB2 for IBM i and DB2 for z/OS servers.

### Examples

The following example creates DB2 databases on a local server:

```
sqldriverconnect 1 0 "attach=true" -3 50 SQL_DRIVER_NOPROMPT
sqlcreatedb 1 sample1 8 null 0 null 0
sqlcreatedb 1 sample2 8 null 0 null 0
```

## SQLCreateDb function (CLI) - Create a database

The following example creates DB2 databases on a remote server:

```
sqldriverconnect 1 0 "attach=true;hostname=myhostname;port=9999;  
uid=myuid;pwd=mypwd;protocol=tcip" -3 50 SQL_DRIVER_NOPROMPT  
sqlcreatedb 1 sample1 8 null 0 null 0  
sqlcreatedb 1 sample2 8 null 0 null 0
```

### Version information

#### Last update

This topic was last updated for IBM DB2 Version 9.7, Fix Pack 3.

#### IBM Data Server Client

Supported in IBM DB2 for Linux, UNIX, and Windows

---

## SQLCreatePkg

SQLCreatePkg() invokes the bind utility, which prepares SQL statements stored in the bind file, and creates a package that is stored in the database.

### Purpose

Specification:	CLI 9.5	-	-
----------------	---------	---	---

### Syntax

```
SQLRETURN SQLCreatePkg(  
    SQLHDBC hDbc,  
    SQLCHAR *szBindFileNameIn,  
    SQLINTEGER cbBindFileNameIn,  
    SQLCHAR *szBindOpts,  
    SQLINTEGER cbBindOpts)
```

### Function arguments

Table 33. SQLCreatePkg() arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hDbc</i>	input	Connection handle.
SQLCHAR*	<i>szBindFileNameIn</i>	input	Name of the file to bind, or the name of a file containing a list of bind file names.
SQLINTEGER	<i>cbBindFileNameIn</i>	input	Number of SQLCHAR elements needed to store <i>szBindFileNameIn</i> , or SQL_NTS if <i>szBindFileNameIn</i> is null-terminated.
SQLCHAR*	<i>szBindOpts</i>	input	List of bind options separated by semicolon.
SQLINTEGER	<i>cbBindOpts</i>	input	Number of SQLCHAR elements needed to store <i>szBindOpts</i> , or SQL_NTS if <i>szBindOpts</i> is null-terminated.

### Usage

The argument *szBindFileNameIn* is a string containing the name of the bind file, or the name of a file containing a list of bind file names. The bind file names must contain the extension `.bnd`. You can specify a path for these files. Precede the name of a bind list file with the at sign (`@`). The following example is a fully qualified bind list file name:

```
/u/user1/sqllib/bnd/@all.lst
```



The bind list file should contain one or more bind file names, and must have the extension `.lst`. Precede all but the first bind file name with a plus symbol (+). The bind file names can be on one or more lines. For example, the bind list file `all.lst` might contain the following lines:

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

You can use path specifications on bind file names in the list file. If no path is specified, the database manager takes path information from the bind list file.

The following **BIND** command parameters can be specified with `SQLCreatePkg()`:

- **KEEPDYNAMIC**={YES | NO}
- **ISOLATION**={CS | NC | RR | RS | UR}
- **BLOCKING**={YES | NO | UNAMBIG}
- **ENCODING**={ASCII | EBCDIC | UNICODE | CCSID | *integer*} (DB2 for z/OS and OS/390® only)
- **REOPT**={NONE | ONCE | ALWAYS}
- **COLLECTION**={schema name}

The **BIND** command parameters can be passed in as a string with name-value pairs separated by a semicolon. For example:

```
keepdynamic=yes; isolation=cs; blocking=no
```

Both options and values are case insensitive.

*Example 1:* Binding a file with **REOPT=ONCE** and **ENCODING=CCSID**

```
strcpy (bindFileName, "insertEmp.bnd");
cliRC = SQLCreatePkg(hdbc,
                    bindFileName,
                    -3, // SQL_NTS
                    "REOPT=ONCE; ENCODING=CCSID");
```

*Example 2:* Binding a list of files all with **KEEPDYNAMIC=YES**, **BLOCKING=NO**, and **ISOLATION=RS**

```
strcpy (bindFileName, "/u/user1/sql1lib/bnd/@all.lst");
cliRC = SQLCreatePkg(hdbc,
                    bindFileName,
                    strlen(bindFileName),
                    "KEEPDYNAMIC=YES; BLOCKING=NO; ISOLATION=RS");
```

*Example 3:* Binding a file with **COLLECTION=SCHEMA NAME**

```
strcpy (bindFileName, "insertEmp.bnd");
cliRC = SQLCreatePkg(hdbc,
                    bindFileName,
                    -3, // SQL_NTS
                    "REOPT=ONCE; ENCODING=CCSID;
                    COLLECTION=NEWTON");
```

---

## SQLDataSources function (CLI) - Get list of data sources

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

## SQLDataSources function (CLI) - Get list of data sources

SQLDataSources() returns a list of target databases available, one at a time. A database must be cataloged to be available.

SQLDataSources() is usually called before a connection is made, to determine the databases that are available to connect to.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLDataSourcesW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLDataSources (
    SQLHENV EnvironmentHandle, /* henv */
    SQLUSMALLINT Direction, /* fDirection */
    SQLCHAR *ServerName, /* *szDSN */
    SQLSMALLINT BufferLength1, /* cbDSNMax */
    SQLSMALLINT *NameLength1Ptr, /* *pcbDSN */
    SQLCHAR *Description, /* *szDescription */
    SQLSMALLINT BufferLength2, /* cbDescriptionMax */
    SQLSMALLINT *NameLength2Ptr); /* *pcbDescription */
```

### Function arguments

Table 34. SQLDataSources arguments

Data type	Argument	Use	Description
SQLHENV	<i>EnvironmentHandle</i>	input	Environment handle.
SQLUSMALLINT	<i>Direction</i>	input	Used by application to request the first data source name in the list or the next one in the list. <i>Direction</i> can take on only the following values: <ul style="list-style-type: none"> <li>SQL_FETCH_FIRST</li> <li>SQL_FETCH_NEXT</li> </ul>
SQLCHAR *	<i>ServerName</i>	output	Pointer to buffer in which to return the data source name.
SQLSMALLINT	<i>BufferLength1</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>ServerName</i> buffer. This number should be less than or equal to SQL_MAX_DSN_LENGTH + 1.
SQLSMALLINT *	<i>NameLength1Ptr</i>	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in <i>ServerName</i> . If the number of SQLCHAR or SQLWCHAR elements available to return is greater than or equal to <i>BufferLength1</i> , the data source name in <i>ServerName</i> is truncated to <i>BufferLength1</i> minus the length of a null-termination character.
SQLCHAR *	<i>Description</i>	output	Pointer to buffer where the description of the data source is returned. CLI will return the <i>Comment</i> field associated with the database cataloged to the DBMS.
SQLSMALLINT	<i>BufferLength2</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>Description</i> buffer.

## SQLDataSources function (CLI) - Get list of data sources

Table 34. SQLDataSources arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	NameLength2Ptr	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in *Description. If the number of SQLCHAR or SQLWCHAR elements available to return is greater than or equal to BufferLength2, the driver description in *Description is truncated to BufferLength2 minus the length of a null-termination character.

### Usage

The application can call this function any time with *Direction* set to either SQL\_FETCH\_FIRST or SQL\_FETCH\_NEXT.

If SQL\_FETCH\_FIRST is specified, the first database in the list will always be returned.

If SQL\_FETCH\_NEXT is specified:

- Directly following a SQL\_FETCH\_FIRST call, the second database in the list is returned
- Before any other SQLDataSources() call, the first database in the list is returned
- When there are no more databases in the list, SQL\_NO\_DATA\_FOUND is returned. If the function is called again, the first database is returned.
- Any other time, the next database in the list is returned.

In an ODBC environment, the ODBC Driver Manager will perform this function.

Since the IBM RDBMSs always returns the description of the data source blank padded to 30 bytes, CLI will do the same.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

### Diagnostics

Table 35. SQLDataSources SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data source name returned in the argument <i>ServerName</i> was longer than the value specified in the argument <i>BufferLength1</i> . The argument <i>NameLength1Ptr</i> contains the length of the full data source name. (Function returns SQL_SUCCESS_WITH_INFO.)  The data source name returned in the argument <i>Description</i> was longer than the value specified in the argument <i>BufferLength2</i> . The argument <i>NameLength2Ptr</i> contains the length of the full data source description. (Function returns SQL_SUCCESS_WITH_INFO.)
58004	Unexpected system failure.	Unrecoverable system error.

## SQLDataSources function (CLI) - Get list of data sources

Table 35. SQLDataSources SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by SQLGetDiagRec() in the <i>MessageText</i> argument describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument <i>BufferLength1</i> was less than 0. The value specified for argument <i>BufferLength2</i> was less than 0.
HY103	Direction option out of range.	The value specified for the argument <i>Direction</i> was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

### Authorization

None.

### Example

```
/* get list of data sources */
cliRC = SQLDataSources(henv,
                      SQL_FETCH_FIRST,
                      dbAliasBuf,
                      SQL_MAX_DSN_LENGTH + 1,
                      &aliasLen,
                      dbCommentBuf,
                      255,
                      &commentLen);
```

## SQLDescribeCol function (CLI) - Return a set of attributes for a column

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLDescribeCol() returns a set of commonly used descriptor information (column name, type, precision, scale, nullability) for the indicated column in the result set generated by a query.

This information is also available in the fields of the IRD.

If the application needs only one attribute of the descriptor information, or needs an attribute not returned by SQLDescribeCol(), the SQLColAttribute() function can be used in place of SQLDescribeCol().

Either SQLPrepare() or SQLExecDirect() must be called before calling this function.

## SQLDescribeCol function (CLI) - Return a set of attributes for a column

This function (or SQLColAttribute()) is usually called before a bind column function (SQLBindCol(), SQLBindFileToCol()) to determine the attributes of a column before binding it to an application variable.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLDescribeColW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLDescribeCol (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLUSMALLINT      ColumnNumber,   /* icol */
    SQLCHAR           *ColumnName,     /* szColName */
    SQLSMALLINT       BufferLength,     /* cbColNameMax */
    SQLSMALLINT       *NameLengthPtr,  /* pcbColName */
    SQLSMALLINT       *DataTypePtr,    /* pfSqlType */
    SQLULEN           *ColumnSizePtr,  /* pcbColDef */
    SQLSMALLINT       *DecimalDigitsPtr, /* piScale */
    SQLSMALLINT       *NullablePtr;    /* pfNullable */
)
```

### Function arguments

Table 36. SQLDescribeCol arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLUSMALLINT	<i>ColumnNumber</i>	input	Column number to be described. Columns are numbered sequentially from left to right, starting at 1. This can also be set to 0 to describe the bookmark column.
SQLCHAR *	<i>ColumnName</i>	output	Pointer to column name buffer. This value is read from the SQL_DESC_NAME field of the IRD. This is set to NULL if the column name cannot be determined.
SQLSMALLINT	<i>BufferLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the * <i>ColumnName</i> buffer.
SQLSMALLINT *	<i>NameLengthPtr</i>	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in * <i>ColumnName</i> . Truncation of column name (* <i>ColumnName</i> ) to <i>BufferLength</i> - 1 SQLCHAR or SQLWCHAR elements occurs if <i>NameLengthPtr</i> is greater than or equal to <i>BufferLength</i> .
SQLSMALLINT *	<i>DataTypePtr</i>	output	Base SQL data type of column. To determine if there is a User Defined Type associated with the column, call SQLColAttribute() with <i>fDescType</i> set to SQL_COLUMN_DISTINCT_TYPE. Refer to the Symbolic SQL Data Type column of the symbolic and default data types table for the data types that are supported.

## SQLDescribeCol function (CLI) - Return a set of attributes for a column

Table 36. SQLDescribeCol arguments (continued)

Data type	Argument	Use	Description
SQLULEN *	<i>ColumnSizePtr</i>	output	Precision of column as defined in the database.  If <i>fSqlType</i> denotes a graphic or DBCLOB SQL data type, then this variable indicates the maximum number of double-byte <i>characters</i> the column can hold.
SQLSMALLINT *	<i>DecimalDigitsPtr</i>	output	Scale of column as defined in the database (only applies to SQL_DECIMAL, SQL_NUMERIC, SQL_TYPE_TIMESTAMP). Refer to the data type scale table for the scale of each of the SQL data types.
SQLSMALLINT *	<i>NullablePtr</i>	output	Indicates whether NULLS are allowed for this column <ul style="list-style-type: none"><li>• SQL_NO_NULLS</li><li>• SQL_NULLABLE</li></ul>

### Usage

Columns are identified by a number, are numbered sequentially from left to right, and can be described in any order.

- Column numbers start at 1 if bookmarks are not used (SQL\_ATTR\_USE\_BOOKMARKS statement attribute set to SQL\_UB\_OFF).
- The *ColumnNumber* argument can be set to 0 to describe the bookmark column if bookmarks are used (the statement attribute is set to SQL\_UB\_ON).

If a null pointer is specified for any of the pointer arguments, CLI assumes that the information is not needed by the application and nothing is returned.

If the column is a User Defined Type, SQLDescribeCol() only returns the built-in type in *DataPtr*. Call SQLColAttribute() with *fDescType* set to SQL\_COLUMN\_DISTINCT\_TYPE to obtain the User Defined Type.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

If SQLDescribeCol() returns either SQL\_ERROR, or SQL\_SUCCESS\_WITH\_INFO, one of the following SQLSTATES can be obtained by calling the SQLGetDiagRec() or SQLGetDiagField() function.

Table 37. SQLDescribeCol SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated.	The column name returned in the argument * <i>ColumnName</i> was longer than the value specified in the argument <i>BufferLength</i> . The argument * <i>NameLengthPtr</i> contains the length of the full column name. (Function returns SQL_SUCCESS_WITH_INFO.)

## SQLDescribeCol function (CLI) - Return a set of attributes for a column

Table 37. SQLDescribeCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07005	The statement did not return a result set.	The statement associated with the <i>StatementHandle</i> did not return a result set. There were no columns to describe. (Call <code>SQLNumResultCols()</code> first to determine if there are any rows in the result set.)
07009	Invalid descriptor index	The value specified for <i>ColumnNumber</i> was equal to 0, and the <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was <code>SQL_UB_OFF</code> . The value specified for the argument <i>ColumnNumber</i> was greater than the number of columns in the result set.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The function was called prior to calling <code>SQLPrepare()</code> or <code>SQLExecuteDirect()</code> for the <i>StatementHandle</i> .  The function was called while in a data-at-execute ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.  The function was called while within a <code>BEGIN COMPOUND</code> and <code>END COMPOUND</code> SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The length specified in argument <i>BufferLength</i> less than 1.
HYC00	Driver not capable.	The SQL data type of column <i>ColumnNumber</i> is not recognized by CLI.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

### Restrictions

The following ODBC defined data types are not supported:

- `SQL_BIT`
- `SQL_TINYINT`

### Example

```

/* return a set of attributes for a column */
cliRC = SQLDescribeCol(hstmt,
                      (SQLSMALLINT)(i + 1),
                      colName,
                      sizeof(colName),
                      &colNameLen,

```

## SQLDescribeCol function (CLI) - Return a set of attributes for a column

```
&colType,
&colSize,
&colScale,
NULL);
```

## SQLDescribeParam function (CLI) - Return description of a parameter marker

### Purpose

Specification:	CLI 5.0	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLDescribeParam() returns the description of a parameter marker associated with a prepared SQL statement. This information is also available in the fields of the IPD. If deferred prepared is enabled, and this is the first call to SQLDescribeParam(), SQLNumResultCols(), or SQLDescribeCol(), the call will force a PREPARE of the SQL statement to be flowed to the server.

### Syntax

```
SQLRETURN SQLDescribeParam (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLUSMALLINT ParameterNumber, /* ipar */
    SQLSMALLINT *DataTypePtr, /* pfSqlType */
    SQLULEN *ParameterSizePtr, /* pcbParamDef */
    SQLSMALLINT *DecimalDigitsPtr, /* pibScale */
    SQLSMALLINT *NullablePtr); /* pfNullable */
```

### Function arguments

Table 38. SQLDescribeParam arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLUSMALLINT	<i>ParameterNumber</i>	input	Parameter marker number ordered sequentially in increasing parameter order, starting at 1.
SQLSMALLINT *	<i>DataTypePtr</i>	output	Pointer to a buffer in which to return the SQL data type of the parameter. This value is read from the SQL_DESC_CONCISE_TYPE record field of the IPD.  When ColumnNumber is equal to 0 (for a bookmark column), SQL_BINARY is returned in <i>*DataTypePtr</i> for variable-length bookmarks.
SQLULEN *	<i>ParameterSizePtr</i>	output	Pointer to a buffer in which to return the size of the column or expression of the corresponding parameter marker as defined by the data source.
SQLSMALLINT *	<i>DecimalDigitsPtr</i>	output	Pointer to a buffer in which to return the number of decimal digits of the column or expression of the corresponding parameter as defined by the data source.



## SQLDescribeParam function (CLI) - Return description of a parameter marker

Table 38. SQLDescribeParam arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	NullablePtr	output	<p>Pointer to a buffer in which to return a value that indicates whether the parameter allows NULL values. This value is read from the SQL_DESC_NULLABLE field of the IPD.</p> <p>One of the following:</p> <ul style="list-style-type: none"> <li>• SQL_NO_NULLS: The parameter does not allow NULL values (this is the default value).</li> <li>• SQL_NULLABLE: The parameter allows NULL values.</li> <li>• SQL_NULLABLE_UNKNOWN: Cannot determine if the parameter allows NULL values.</li> </ul> <p><b>Note:</b> The CLI driver returns SQL_NULLABLE_UNKNOWN.</p>

### Usage

Parameter markers are numbered in increasing order as they appear in the SQL statement, starting with 1.

SQLDescribeParam() does not return the type (input, input/output, or output) of a parameter in an SQL statement. Except in calls to stored procedures, all parameters in SQL statements are input parameters. To determine the type of each parameter in a call to a stored procedure, call SQLProcedureColumns().

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 39. SQLDescribeParam SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index.	<p>The value specified for the argument <i>ParameterNumber</i> less than 1.</p> <p>The value specified for the argument <i>ParameterNumber</i> was greater than the number of parameters in the associated SQL statement.</p> <p>The parameter marker was part of a non-DML statement.</p> <p>The parameter marker was part of a SELECT list.</p>
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
21S01	Insert value list does not match column list.	The number of parameters in the INSERT statement did not match the number of columns in the table named in the statement.

## SQLDescribeParam function (CLI) - Return description of a parameter marker

Table 39. SQLDescribeParam SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The function was called prior to calling SQLPrepare() or SQLExecDirect() for the <i>StatementHandle</i> .  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.  SQLExecute(), SQLExecDirect(), SQLBulkOperations(), or SQLSetPos() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYC00	Driver not capable.	The schema function stored procedures are not accessible on the server. Install the schema function stored procedures on the server and ensure they are accessible.

### Restrictions

None.

## SQLDisconnect function (CLI) - Disconnect from a data source

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLDisconnect() closes the connection associated with the database connection handle.

SQLEndTran() must be called before calling SQLDisconnect() if an outstanding transaction exists on this connection.

After calling this function, either call SQLConnect() to connect to another database, or use SQLFreeHandle() to free the connection handle.

### Syntax

```
SQLRETURN SQLDisconnect (SQLHDBC ConnectionHandle;) /* hdbc */
```

### Function arguments

Table 40. SQLDisconnect arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle

### Usage

If an application calls `SQLDisconnect()` before it has freed all the statement handles associated with the connection, CLI frees them after it successfully disconnects from the database.

If `SQL_SUCCESS_WITH_INFO` is returned, it implies that even though the disconnect from the database is successful, additional error or implementation specific information is available. For example, a problem was encountered on the clean up subsequent to the disconnect, or if there is no current connection because of an event that occurred independently of the application (such as communication failure).

After a successful `SQLDisconnect()` call, the application can re-use *ConnectionHandle* to make another `SQLConnect()` or `SQLDriverConnect()` request.

An application should not rely on `SQLDisconnect()` to close cursors (with both stored procedures and regular client applications). In both cases the cursor should be closed using `SQLCloseCursor()`, then the statement handle freed using `SQLFreeHandle()`.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 41. SQLDisconnect SQLSTATEs

SQLSTATE	Description	Explanation
01002	Disconnect error.	An error occurred during the disconnect. However, the disconnect succeeded. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection is closed.	The connection specified in the argument <i>ConnectionHandle</i> was not open.
25000 25501	Invalid transaction state.	There was a transaction in process on the connection specified by the argument <i>ConnectionHandle</i> . The transaction remains active, and the connection cannot be disconnected. <b>Note:</b> This error does not apply to stored procedures written in CLI.
25501	Invalid transaction state.	There was a transaction in process on the connection specified by the argument <i>ConnectionHandle</i> . The transaction remains active, and the connection cannot be disconnected.
58004	Unexpected system failure.	Unrecoverable system error.

## SQLDisconnect function (CLI) - Disconnect from a data source

Table 41. SQLDisconnect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

### Restrictions

None.

### Example

```
SQLHANDLE hdbc; /* connection handle */  
  
/* ... */  
  
/* disconnect from the database */  
cliRC = SQLDisconnect(hdbc);
```

## SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

SQLDriverConnect() is an alternative to SQLConnect(). Both functions establish a connection to the target database, but SQLDriverConnect() supports additional connection parameters and the ability to prompt the user for connection information.

Use SQLDriverConnect() when the data source requires parameters other than the 3 input arguments supported by SQLConnect() (data source name, user ID and password), or when you want to use CLI's graphical user interface to prompt the user for mandatory connection information.

Once a connection is established, the completed connection string is returned. Applications can store this string for future connection requests.

### Syntax

#### Generic

```
SQLRETURN SQLDriverConnect (  
    SQLHDBC      ConnectionHandle,           /* hdbc */  
    SQLHWND      WindowHandle,              /* hwnd */  
    SQLCHAR      *InConnectionString,      /* szConnStrIn */  
    SQLSMALLINT  InConnectionStringLength,  /* cbConnStrIn */  
    SQLCHAR      *OutConnectionString,      /* szConnStrOut */
```

## SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

```

SQLSMALLINT OutConnectionStringCapacity, /* cbConnStrOutMax */
SQLSMALLINT *OutConnectionStringLengthPtr, /* pcbConnStrOut */
SQLUSMALLINT DriverCompletion); /* fDriverCompletion */

```

### Function arguments

Table 42. SQLDriverConnect arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle
SQLHWND	<i>WindowHandle</i>	input	Window handle. On the Windows platform, this is the parent Windows handle. Currently the window handle is only supported on Windows.  If a NULL is passed, then no dialog will be presented.
SQLCHAR *	<i>InConnectionString</i>	input	A full, partial or empty (null pointer) connection string (see syntax and description below).
SQLSMALLINT	<i>InConnectionStringLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>InConnectionString</i> .
SQLSMALLINT *	<i>OutConnectionString</i>	output	Pointer to buffer for the completed connection string.  If the connection was established successfully, this buffer will contain the completed connection string. Applications should allocate at least SQL_MAX_OPTION_STRING_LENGTH bytes for this buffer.
SQLSMALLINT	<i>OutConnectionStringCapacity</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>OutConnectionString</i> .
SQLSMALLINT *	<i>OutConnectionStringLengthPtr</i>	output	Pointer to the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in the <i>OutConnectionString</i> buffer.  If the value of <i>*OutConnectionStringLengthPtr</i> is greater than or equal to <i>OutConnectionStringCapacity</i> , the completed connection string in <i>OutConnectionString</i> is truncated to <i>OutConnectionStringCapacity</i> - 1 SQLCHAR or SQLWCHAR elements.
SQLUSMALLINT	<i>DriverCompletion</i>	input	Indicates when CLI should prompt the user for more information.  Possible values: <ul style="list-style-type: none"> <li>• SQL_DRIVER_PROMPT</li> <li>• SQL_DRIVER_COMPLETE</li> <li>• SQL_DRIVER_COMPLETE_REQUIRED</li> <li>• SQL_DRIVER_NOPROMPT</li> </ul>

### Usage

#### InConnectionString Argument

A request connection string has the following syntax:

## SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

connection-string ::= attribute[;] | attribute; connection-string

attribute ::= attribute-keyword=attribute-value  
| DRIVER=[{attribute-value}]

attribute-keyword ::= DSN | UID | PWD | NEWPWD  
| driver-defined-attribute-keyword

attribute-value ::= character-string  
driver-defined-attribute-keyword ::= identifier

where

- character-string has zero or more SQLCHAR or SQLWCHAR elements
- identifier has one or more SQLCHAR or SQLWCHAR elements
- attribute-keyword is case insensitive
- attribute-value may be case sensitive
- the value of the **DSN** keyword does not consist solely of blanks
- **NEWPWD** is used as part of a change password request. The application can either specify the new string to use, for example, **NEWPWD=anewpass**; or specify **NEWPWD=**; and rely on a dialog box generated by the CLI driver to prompt for the new password

Because of connection string and initialization file grammar, keywords and attribute values that contain the characters `[]{}(),;?*=@` should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (`\`) character. For CLI Version 2, braces are required around the **DRIVER** keyword.

If any keywords are repeated in the browse request connection string, CLI uses the value associated with the first occurrence of the keyword. If the **DSN** and **DRIVER** keywords are included in the same browse request connection string, CLI uses whichever keyword appears first.

### OutConnectionString Argument

The result connection string is a list of connection attributes. A connection attribute consists of an attribute keyword and a corresponding attribute value. The browse result connection string has the following syntax:

connection-string ::= attribute[;] | attribute; connection-string

attribute ::= [\*]attribute-keyword=attribute-value  
attribute-keyword ::= ODBC-attribute-keyword  
| driver-defined-attribute-keyword

ODBC-attribute-keyword = {UID | PWD}[:localized-identifier]  
driver-defined-attribute-keyword ::= identifier[:localized-identifier]

attribute-value ::= {attribute-value-list} | ?  
(The braces are literal; they are returned by CLI.)  
attribute-value-list ::= character-string [:localized-character  
string] | character-string [:localized-character string], attribute-value-list

where

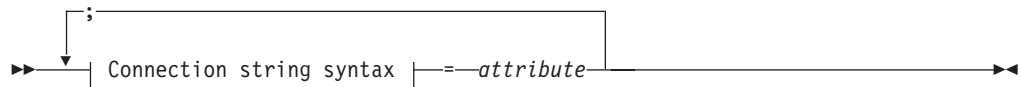
- character-string and localized-character string have zero or more SQLCHAR or SQLWCHAR elements

## SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

- identifier and localized-identifier have one or more SQLCHAR or SQLWCHAR elements; attribute-keyword is case insensitive
- attribute-value may be case sensitive

Because of connection string and initialization file grammar, keywords, localized identifiers, and attribute values that contain the characters [{}() ,;?\*=!@ should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (\) character.

The connection string is used to pass one or more values needed to complete a connection. The contents of the connection string and the value of *DriverCompletion* will determine if CLI needs to establish a dialog with the user.



### Connection string syntax



Each keyword above has an attribute that is equal to the following:

**DSN** Data source name. The name or alias-name of the database. Required if *DriverCompletion* is equal to SQL\_DRIVER\_NOPROMPT.

**UID** Authorization-name (user identifier).

**PWD** The password corresponding to the authorization name. If there is no password for the user ID, an empty value is specified (PWD=;).

#### **NEWPWD**

New password used as part of a change password request. The application can either specify the new string to use, for example, NEWPWD=newpass; or specify NEWPWD=; and rely on a dialog box generated by the CLI driver to prompt for the new password (set the *DriverCompletion* argument to anything other than SQL\_DRIVER\_NOPROMPT).

Any one of the CLI keywords can be specified on the connection string. If any keywords are repeated in the connection string, the value associated with the first occurrence of the keyword is used.

If any keywords exists in the CLI initialization file, the keywords and their respective values are used to augment the information passed to CLI in the connection string. If the information in the CLI initialization file contradicts information in the connection string, the values in connection string take precedence.

If the end user  *Cancels*  a dialog box presented, SQL\_NO\_DATA\_FOUND is returned.

The following values of *DriverCompletion* determines when a dialog will be opened:

## SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

### SQL\_DRIVER\_PROMPT:

A dialog is always initiated. The information from the connection string and the CLI initialization file are used as initial values, to be supplemented by data input via the dialog box.

### SQL\_DRIVER\_COMPLETE:

A dialog is only initiated if there is insufficient information in the connection string. The information from the connection string is used as initial values, to be supplemented by data entered via the dialog box.

### SQL\_DRIVER\_COMPLETE\_REQUIRED:

A dialog is only initiated if there is insufficient information in the connection string. The information from the connection string is used as initial values. Only mandatory information is requested. The user is prompted for required information only.

### SQL\_DRIVER\_NOPROMPT:

The user is not prompted for any information. A connection is attempted with the information contained in the connection string. If there is not enough information, `SQL_ERROR` is returned.

Once a connection is established, the complete connection string is returned. Applications that need to set up multiple connections to the same database for a given user ID should store this output connection string. This string can then be used as the input connection string value on future `SQLDriverConnect()` calls.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLDriverConnectW()`. Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_NO_DATA_FOUND`
- `SQL_INVALID_HANDLE`
- `SQL_ERROR`

### Diagnostics

All of the diagnostics generated by `SQLConnect()` can be returned here as well. The following table shows the additional diagnostics that can be returned.

Table 43. `SQLDriverConnect` `SQLSTATEs`

SQLSTATE	Description	Explanation
01004	Data truncated.	The buffer <code>szConnstrOut</code> was not large enough to hold the entire connection string. The argument <code>*OutConnectionStringLengthPtr</code> contains the actual length of the connection string available for return. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> )
01S00	Invalid connection string attribute.	An invalid keyword or attribute value was specified in the input connection string, but the connection to the data source was successful anyway because one of the following occurred: <ul style="list-style-type: none"><li>• The unrecognized keyword was ignored.</li><li>• The invalid attribute value was ignored, the default value was used instead.</li></ul> (Function returns <code>SQL_SUCCESS_WITH_INFO</code> )



## SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

Table 43. SQLDriverConnect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	The information specified in the connection string was insufficient for making a connect request, but the dialog was prohibited by setting <i>fCompletion</i> to SQL_DRIVER_NOPROMPT.
	Dialog Failed	The attempt to display the dialog failed.
HY090	Invalid string or buffer length.	The value specified for <i>InConnectionStringLength</i> was less than 0, but not equal to SQL_NTS.
		The value specified for <i>OutConnectionStringCapacity</i> was less than 0.
HY110	Invalid driver completion.	The value specified for the argument <i>fCompletion</i> was not equal to one of the valid values.

### Restrictions

None.

### Example

```
rc = SQLDriverConnect(hdbc,  
                      (SQLHWND)sqlHWND,  
                      InConnectionString,  
                      InConnectionStringLength,  
                      OutConnectionString,  
                      OutConnectionStringCapacity,  
                      StrLength2,  
                      DriveCompletion);
```

## SQLDropDb function (CLI) - Drop a database

### Purpose

Specification:	CLI V9.7		
----------------	----------	--	--

The SQLDropDb() function drops the specified database.

**Unicode Equivalent:** The corresponding Unicode function is the SQLDropDbW() function. For information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQL_API_FN SQLDropDb ( SQLHDBC      hDbc,  
                                  SQLCHAR      *szDbName,  
                                  SQLINTEGER   cbDbName);
```

### Function arguments

Table 44. SQLDropDb function argument

Data type	Argument	Use	Description
SQLHDBC	<i>hDbc</i>	input	Connection handle.
SQLCHAR *	<i>szDbName</i>	input	Name of the database that is to be dropped.

## SQLDropDb function (CLI) - Drop a database

Table 44. SQLDropDb function argument (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>cbDbName</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of the function) that is needed to store the <i>szDbName</i> argument or to store SQL_NTS if the <i>szDbName</i> is null terminated.

### Usage

To drop a DB2 database, the CLI application must first attach to the server instance by using the ATTACH keyword. The valid APIs, after connecting to the server instance using ATTACH keyword are SQLCreateDb(), SQLDropDb(), and SQLDisconnect().

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR

### Restrictions

- An already connected database cannot be dropped.
- The SQLDropDb() function is not supported for DB2 for IBM i and DB2 for z/OS data servers.

### Example

The following example creates and drops DB2 databases on a local server:

```
sqldriverconnect 1 0 "attach=true" -3 50 SQL_DRIVER_NOPROMPT
sqlcreatedb 1 sample1 8 null 0 null 0
sqlcreatedb 1 sample2 8 null 0 null 0
sqldropdb 1 sample1 8
sqldropdb 1 sample2 8
sqldisconnect 1
```

The following example creates and drops DB2 databases on a remote server:

```
sqldriverconnect 1 0 "attach=true;hostname=myhostname;port=9999;
uid=myuid;pwd=mypwd;protocol=tcip" -3 50 SQL_DRIVER_NOPROMPT
sqlcreatedb 1 sample1 8 null 0 null 0
sqlcreatedb 1 sample2 8 null 0 null 0
sqldropdb 1 sample1 8
sqldropdb 1 sample2 8
sqldisconnect 1
```

### Version information

#### Last update

This topic was last updated for IBM DB2 Version 9.7, Fix Pack 3.

#### IBM Data Server Client

Supported in IBM DB2 Database for Linux, UNIX, and Windows

---

## SQLEndTran function (CLI) - End transactions of a connection or an environment

Requests a commit or rollback operation for all active operations on all statements associated with a connection, or for all connections associated with an environment.

## SQLEndTran function (CLI) - End transactions of a connection or an environment

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLEndTran() requests a commit or rollback operation for all active operations on all statements that are associated with a connection, or for all connections that are associated with an environment.

### Syntax

```
SQLRETURN SQLEndTran (
    SQLSMALLINT HandleType, /* fHandleType */
    SQLHANDLE Handle, /* hHandle */
    SQLSMALLINT CompletionType); /* fType */
```

### Function arguments

Table 45. SQLEndTran arguments

Data type	Argument	Use	Description
SQLSMALLINT	<i>HandleType</i>	Input	The <i>HandleType</i> type identifier contains either SQL_HANDLE_ENV if <i>Handle</i> is an environment handle, or SQL_HANDLE_DBC if <i>Handle</i> is a connection handle.
SQLHANDLE	<i>Handle</i>	Input	The handle, of the type that is indicated by <i>HandleType</i> , that indicates the scope of the transaction.
SQLSMALLINT	<i>CompletionType</i>	Input	One of the following two values: <ul style="list-style-type: none"><li>• SQL_COMMIT</li><li>• SQL_ROLLBACK</li></ul>

### Usage

If *HandleType* is SQL\_HANDLE\_ENV and *Handle* is a valid environment handle, CLI attempts to commit or roll back transactions one at a time, depending on the value of *CompletionType*, on all connections that are in a connected state on that environment. SQL\_SUCCESS is returned only if it receives SQL\_SUCCESS for each connection. If it receives SQL\_ERROR on one or more connections, it returns SQL\_ERROR to the application, and the diagnostic information is placed in the diagnostic data structure of the environment. To determine which connections failed during the commit or rollback operation, the application can call SQLGetDiagRec() for each connection.

You should not use SQLEndTran() when working in a Distributed Unit of Work environment. Use the transaction manager APIs instead.

If *CompletionType* is SQL\_COMMIT, SQLEndTran() issues a commit request for all active operations on any statement that is associated with an affected connection. If *CompletionType* is SQL\_ROLLBACK, SQLEndTran() issues a rollback request for all active operations on any statement that is associated with an affected connection. If no transactions are active, SQLEndTran() returns SQL\_SUCCESS with no effect on any data sources.

To determine how transaction operations affect cursors, an application calls SQLGetInfo() with the SQL\_CURSOR\_ROLLBACK\_BEHAVIOR and SQL\_CURSOR\_COMMIT\_BEHAVIOR options.

## SQLEndTran function (CLI) - End transactions of a connection or an environment

If the `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` value equals `SQL_CB_DELETE`, `SQLEndTran()` closes and deletes all open cursors on all statements that are associated with the connection, and discards all pending results. `SQLEndTran()` leaves any statement present in an allocated (unprepared) state; the application can reuse them for subsequent SQL requests or can call `SQLFreeStmt()` or `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_STMT` to deallocate them.

If the `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` value equals `SQL_CB_CLOSE`, `SQLEndTran()` closes all open cursors on all statements that are associated with the connection. `SQLEndTran()` leaves any statement present in a prepared state; the application can call `SQLExecute()` for a statement that is associated with the connection without first calling `SQLPrepare()`.

If the `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` value equals `SQL_CB_PRESERVE`, `SQLEndTran()` does not affect open cursors that are associated with the connection. Cursors remain at the row that they pointed to prior to the call to `SQLEndTran()`.

When autocommit mode is off, calling `SQLEndTran()` with either `SQL_COMMIT` or `SQL_ROLLBACK` when no transaction is active returns `SQL_SUCCESS`, which indicates that there is no work to be committed or rolled back. Calling `SQLEndTran()` has no effect on the data source, unless errors that are not related to the transactions occur.

When autocommit mode is on, calling `SQLEndTran()` with a *CompletionType* of either `SQL_COMMIT` or `SQL_ROLLBACK` always returns `SQL_SUCCESS`, unless errors that are not related to the transactions occur.

When a CLI application is running in autocommit mode, the CLI driver does not pass the statement to the server.

For applications that use the ODBC driver version 3.8 or later, the `SQLEndTran` function can set the connection to suspended state and returns `SQL_ERROR` (with `SQLSTATE` set to `HY117`). You must set the `SQL_ATTR_ODBC_VERSION` environment attribute to `SQL_OV_ODBC3_80`. For more details about necessary conditions to set the connection in a suspended state, see the Microsoft MSDN documentation for the `SQLEndTran()` at [http://msdn.microsoft.com/en-us/library/ms716544\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms716544(v=vs.85).aspx).

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 46. `SQLEndTran` `SQLSTATEs`

SQLSTATE	Description	Explanation
01000	Warning.	An informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection is closed.	The <i>ConnectionHandle</i> is not in a connected state.

## SQLEndTran function (CLI) - End transactions of a connection or an environment

Table 46. *SQLEndTran SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
08007	Connection failure during transaction.	The connection that is associated with the <i>ConnectionHandle</i> failed during the execution of the function, and it cannot be determined whether the requested <b>COMMIT</b> or <b>ROLLBACK</b> occurred before the failure.
40001	Transaction rollback.	The transaction is rolled back due to a resource deadlock with another transaction.
HY000	General error.	An error occurred for which there is no specific SQLSTATE. The error message returned by <i>SQLGetDiagRec()</i> in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	<p>An asynchronously executing function was called for a <i>StatementHandle</i> that is associated with the <i>ConnectionHandle</i> and was still executing when <i>SQLEndTran()</i> was called.</p> <p><i>SQLExecute()</i> or <i>SQLExecDirect()</i> was called for a <i>StatementHandle</i> that is associated with the <i>ConnectionHandle</i> and returned <i>SQL_NEED_DATA</i>. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>An exception to this behavior exists for CLI applications that run against a DB2 for z/OS database server. When the connection attribute <i>SQL_ATTR_FORCE_ROLLBACK</i> is turned on, CLI applications can successfully perform <i>SQLEndTran()</i> or <i>SQLTransact()</i> when <i>CompletionType</i> is <i>SQL_ROLLBACK</i>. The <i>StreamPutData</i> configuration keyword must be set to 1 (on).</p>
HY012	Invalid transaction code.	The value that is specified for the argument <i>CompletionType</i> is neither <i>SQL_COMMIT</i> nor <i>SQL_ROLLBACK</i> .
HY092	Option type out of range.	The value specified for the argument <i>HandleType</i> was neither <i>SQL_HANDLE_ENV</i> nor <i>SQL_HANDLE_DBC</i> .

### Restrictions

None.

### Example

```
/* commit all active transactions on the connection */
cliRC = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT)

/* ... */

/* rollback all active transactions on the connection */
cliRC = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK);

/* ... */

/* rollback all active transactions on all connections
   in this environment */
cliRC = SQLEndTran(SQL_HANDLE_ENV, henv, SQL_ROLLBACK);
```

---

### SQLError function (CLI) - Retrieve error information

#### Deprecated

##### Note:

In ODBC 3.0, SQLError() has been deprecated and replaced with SQLGetDiagRec() and SQLGetDiagField().

Although this version of CLI continues to support SQLError(), use SQLGetDiagRec() in your CLI programs so that they conform to the latest standards.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLErrorW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

#### Migrating to the new function

To read the error diagnostic records for a statement handle, the SQLError() function,

```
SQLError(henv, hdbc, hstmt, *szSqlState, *pfNativeError,  
        *szErrorMsg, cbErrorMsgMax, *pcbErrorMsg);
```

for example, would be rewritten using the new function as:

```
SQLGetDiagRec(SQL_HANDLE_HSTMT, hstmt, 1, szSqlState, pfNativeError,  
             szErrorMsg, cbErrorMsgMax, pcbErrorMsg);
```

---

### SQLExecDirect function (CLI) - Execute a statement directly

#### Purpose

<b>Specification:</b>	CLI 1.1	ODBC 1.0	ISO CLI
-----------------------	---------	----------	---------

SQLExecDirect() directly executes the specified SQL statement or XQuery expression using the current values of the parameter marker variables if any parameters exist in the statement. The statement or expression can only be executed once.

For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the XMLQUERY function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in XMLQUERY for execution.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLExecDirectW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

#### Syntax

```
SQLRETURN SQLExecDirect (  
        SQLHSTMT      StatementHandle,      /* hstmt */  
        SQLCHAR       *StatementText,      /* szSqlStr */  
        SQLINTEGER     TextLength);      /* cbSqlStr */
```

## Function arguments

Table 47. SQLExecDirect arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle. There must not be an open cursor associated with <i>StatementHandle</i> .
SQLCHAR *	<i>StatementText</i>	input	SQL statement or XQuery expression string.
SQLINTEGER	<i>TextLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>StatementText</i> argument, or SQL_NTS if <i>StatementText</i> is null-terminated.

## Usage

If the SQL statement text contains vendor escape clause sequences, CLI will first modify the SQL statement text to the appropriate DB2-specific format before submitting it for preparation and execution. If the application does not generate SQL statements that contain vendor escape clause sequences, then it should set the SQL\_ATTR\_NOSCAN statement attribute to SQL\_NOSCAN\_ON at the connection level so that CLI does not perform a scan for vendor escape clauses.

The SQL statement can be COMMIT or ROLLBACK if it is called using SQLExecDirect(). Doing so yields the same result as calling SQLEndTran() on the current connection handle.

The SQL statement string can contain parameter markers, however all parameters must be bound before calling SQLExecDirect().

If the SQL statement is a query, or *StatementText* is an XQuery expression, SQLExecDirect() will generate a cursor name, and open the cursor. If the application has used SQLSetCursorName() to associate a cursor name with the statement handle, CLI associates the application generated cursor name with the internally generated one.

If a result set is generated, SQLFetch() or SQLFetchScroll() will retrieve the next row (or rows) of data into bound variables, LOB locators, or LOB file references.

If the SQL statement is a positioned DELETE or a positioned UPDATE, the cursor referenced by the statement must be positioned on a row and must be defined on a separate statement handle under the same connection handle.

There must not already be an open cursor on the statement handle.

If SQLSetStmtAttr() has been called with the SQL\_ATTR\_PARAMSET\_SIZE attribute to specify that an array of input parameter values has been bound to each parameter marker, then the application needs to call SQLExecDirect() only once to process the entire array of input parameter values.

If the executed statement returns multiple result sets (one for each set of input parameters), then SQLMoreResults() should be used to advance to the next result set once processing on the current result set is complete.

## SQLExecDirect function (CLI) - Execute a statement directly

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NEED\_DATA
- SQL\_NO\_DATA\_FOUND

SQL\_NEED\_DATA is returned when the application has requested to input data-at-execute parameter values by setting the *\*StrLen\_or\_IndPtr* value specified during SQLBindParameter() to SQL\_DATA\_AT\_EXEC for one or more parameters.

SQL\_NO\_DATA\_FOUND is returned if the SQL statement is a Searched UPDATE or Searched DELETE and no rows satisfy the search condition.

### Diagnostics

Table 48. SQLExecDirect SQLSTATES

SQLSTATE	Description	Explanation
01504	The UPDATE or DELETE statement does not include a WHERE clause.	<i>StatementText</i> contained an UPDATE or DELETE statement which did not contain a WHERE clause. (Function returns SQL_SUCCESS_WITH_INFO or SQL_NO_DATA_FOUND if there were no rows in the table).
01508	Statement disqualified for blocking.	The statement was disqualified for blocking for reasons other than storage.
07001	Wrong number of parameters.	The number of parameters bound to application variables using SQLBindParameter() was less than the number of parameter markers in the SQL statement contained in the argument <i>StatementText</i> .
07006	Invalid conversion.	Transfer of data between CLI and the application variables would result in an incompatible data conversion.
21S01	Insert value list does not match column list.	<i>StatementText</i> contained an INSERT statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degrees of derived table does not match column list.	<i>StatementText</i> contained a CREATE VIEW statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22001	String data right truncation.	A character string assigned to a character type column exceeded the maximum length of the column.
22003	Numeric value out of range.	A numeric value assigned to a numeric type column caused truncation of the whole part of the number, either at the time of assignment or in computing an intermediate result.  <i>StatementText</i> contained an SQL statement with an arithmetic expression which caused division by zero. <b>Note:</b> as a result the cursor state is undefined for DB2 Database for Linux, UNIX, and Windows (the cursor will remain open for other RDBMSs).



## SQLExecDirect function (CLI) - Execute a statement directly

Table 48. *SQLExecDirect SQLSTATES (continued)*

SQLSTATE	Description	Explanation
22005	Error in assignment.	<p><i>StatementText</i> contained an SQL statement with a parameter or literal and the value or LOB locator was incompatible with the data type of the associated table column.</p> <p>The length associated with a parameter value (the contents of the <i>pcbValue</i> buffer specified on <code>SQLBindParameter()</code>) is not valid.</p> <p>The argument <i>fsQLType</i> used in <code>SQLBindParameter()</code> or <code>SQLSetParam()</code>, denoted an SQL graphic data type, but the deferred length argument (<i>pcbValue</i>) contains an odd length value. The length value must be even for graphic data types.</p>
22007	Invalid datetime format.	<i>StatementText</i> contained an SQL statement with an invalid datetime format; that is, an invalid string representation or value was specified, or the value was an invalid date, time, or timestamp.
22008	Datetime field overflow.	Datetime field overflow occurred; for example, an arithmetic operation on a date or timestamp has a result that is not within the valid range of dates, or a datetime value cannot be assigned to a bound variable because it is too small.
22012	Division by zero is invalid.	<i>StatementText</i> contained an SQL statement with an arithmetic expression that caused division by zero.
23000	Integrity constraint violation.	The execution of the SQL statement is not permitted because the execution would cause integrity constraint violation in the DBMS.
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
24504	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row.	Results were pending on the <i>StatementHandle</i> from a previous query or a cursor associated with the <i>hstmt</i> had not been closed.
34000	Invalid cursor name.	<i>StatementText</i> contained a Positioned DELETE or a Positioned UPDATE and the cursor referenced by the statement being executed was not open.
37xxx <sup>a</sup>	Invalid SQL syntax.	<p><i>StatementText</i> contained one or more of the following:</p> <ul style="list-style-type: none"> <li>• an SQL statement that the connected database server could not prepare</li> <li>• a statement containing a syntax error</li> </ul>
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42xxx	Syntax Error or Access Rule Violation.	<p>425xx indicates the authorization ID does not have permission to execute the SQL statement contained in <i>StatementText</i>.</p> <p>Other 42xxx SQLSTATES indicate a variety of syntax or access problems with the statement.</p>

## SQLExecDirect function (CLI) - Execute a statement directly

Table 48. SQLExecDirect SQLSTATES (continued)

SQLSTATE	Description	Explanation
428A1	Unable to access a file referenced by a host file variable.	<p>This can be raised for any of the following scenarios. The associated reason code in the text identifies the particular error:</p> <ul style="list-style-type: none"> <li>• 01 - The file name length is invalid, or the file name, the path has an invalid format, or both.</li> <li>• 02 - The file option is invalid. It must have one of the following values: <ul style="list-style-type: none"> <li>SQL_FILE_READ -read from an existing file</li> <li>SQL_FILE_CREATE -create a new file for write</li> <li>SQL_FILE_OVERWRITE -overwrite an existing file. If the file does not exist, create the file.</li> <li>SQL_FILE_APPEND -append to an existing file. If the file does not exist, create the file.</li> </ul> </li> <li>• 03 - The file cannot be found.</li> <li>• 04 - The SQL_FILE_CREATE option was specified for a file with the same name as an existing file.</li> <li>• 05 - Access to the file was denied. The user does not have permission to open the file.</li> <li>• 06 - Access to the file was denied. The file is in use with incompatible modes. Files to be written to are opened in exclusive mode.</li> <li>• 07 - Disk full was encountered while writing to the file.</li> <li>• 08 - Unexpected end of file encountered while reading from the file.</li> <li>• 09 - A media error was encountered while accessing the file.</li> </ul>
42895	The value of a host variable in the EXECUTE or OPEN statement cannot be used because of its data type.	<p>The LOB locator type specified on the bind parameter function call does not match the LOB data type of the parameter marker.</p> <p>The argument <i>fsQLType</i> used on the bind parameter function specified a LOB locator type but the corresponding parameter marker is not a LOB.</p>
44000	Integrity constraint violation.	<i>StatementText</i> contained an SQL statement which contained a parameter or literal. This parameter value was NULL for a column defined as NOT NULL in the associated table column, or a duplicate value was supplied for a column constrained to contain only unique values, or some other integrity constraint was violated.
56084	LOB data is not supported in DRDA®.	LOB columns cannot either be selected or updated when connecting to host or AS/400® servers (using DB2 Connect™).
58004	Unexpected system failure.	Unrecoverable system error.
S0001	Database object already exists.	<i>StatementText</i> contained a CREATE TABLE or CREATE VIEW statement and the table name or view name specified already existed.
S0002	Database object does not exist.	<i>StatementText</i> contained an SQL statement that references a table name or view name which does not exist.
S0011	Index already exists.	<i>StatementText</i> contained a CREATE INDEX statement and the specified index name already existed.
S0012	Index not found.	<i>StatementText</i> contained a DROP INDEX statement and the specified index name did not exist.

## SQLExecDirect function (CLI) - Execute a statement directly

Table 48. SQLExecDirect SQLSTATES (continued)

SQLSTATE	Description	Explanation
S0021	Column already exists.	<i>StatementText</i> contained an ALTER TABLE statement and the column specified in the ADD clause was not unique or identified an existing column in the base table.
S0022	Column not found.	<i>StatementText</i> contained an SQL statement that references a column name which does not exist.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	<i>StatementText</i> was a null pointer.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The argument <i>TextLength</i> was less than 1 but not equal to SQL_NTS.
HY092	Option type out of range.	The <i>FileOptions</i> argument of a previous SQLBindFileToParam() operation was not valid.
HY503	Invalid file name length.	The <i>fileNameLength</i> argument value from SQLBindFileToParam() was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Note:

**a** xxx refers to any SQLSTATE with that class code. Example, 37xxx refers to any SQLSTATE in the 37 class.

## Restrictions

None.

## Example

```
/* directly execute a statement - end the COMPOUND statement */  
cliRC = SQLExecDirect(hstmt, (SQLCHAR *)"SELECT * FROM ORG", SQL_NTS);
```

## SQLExecute function (CLI) - Execute a statement

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLExecute() executes a statement that was successfully prepared using SQLPrepare() on the same statement handle, once or multiple times. The statement is executed using the current values of any application variables that were bound to parameter markers by SQLBindParameter() or SQLBindFileToParam().

### Syntax

```
SQLRETURN SQLExecute (SQLHSTMT StatementHandle); /* hstmt */
```

## SQLExecute function (CLI) - Execute a statement

### Function arguments

Table 49. SQLExecute arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle. There must not be an open cursor associated with <i>StatementHandle</i> .

### Usage

The SQL statement string previously prepared on *StatementHandle* using `SQLPrepare()` may contain parameter markers. All parameters must be bound before calling `SQLExecute()`.

**Note:** For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the `XMLQUERY` function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in `XMLQUERY` for execution.

Once the application has processed the results from the `SQLExecute()` call, it can execute the statement again with new (or the same) parameter values.

A statement executed by `SQLExecuteDirect()` cannot be re-executed by calling `SQLExecute()`. Only statements prepared with `SQLPrepare()` can be executed and re-executed with `SQLExecute()`.

If the prepared SQL statement is a query or an XQuery expression, `SQLExecute()` will generate a cursor name, and open the cursor. If the application has used `SQLSetCursorName()` to associate a cursor name with the statement handle, CLI associates the application generated cursor name with the internally generated one.

To execute a query more than once on a given statement handle, the application must close the cursor by calling `SQLCloseCursor()` or `SQLFreeStmt()` with the `SQL_CLOSE` option. There must not be an open cursor on the statement handle when calling `SQLExecute()`.

If a result set is generated, `SQLFetch()` or `SQLFetchScroll()` will retrieve the next row (or rows) of data into bound variables, LOB locators or LOB file references.

If the SQL statement is a positioned DELETE or a positioned UPDATE, the cursor referenced by the statement must be positioned on a row at the time `SQLExecute()` is called, and must be defined on a separate statement handle under the same connection handle.

If `SQLSetStmtAttr()` has been called with the `SQL_ATTR_PARAMSET_SIZE` attribute to specify that an array of input parameter values has been bound to each parameter marker, the application needs to call `SQLExecute()` only once to process the entire array of input parameter values. If the executed statement returns multiple result sets (one for each set of input parameters), then `SQLMoreResults()` should be used to advance to the next result set once processing on the current result set is complete.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`

## SQLExecute function (CLI) - Execute a statement

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NEED\_DATA
- SQL\_NO\_DATA\_FOUND

SQL\_NEED\_DATA is returned when the application has requested to input data-at-execute parameter values by setting the *\*StrLen\_or\_IndPtr* value specified during `SQLBindParameter()` to `SQL_DATA_AT_EXEC` for one or more parameters.

SQL\_NO\_DATA\_FOUND is returned if the SQL statement is a searched UPDATE or searched DELETE and no rows satisfy the search condition.

### Diagnostics

The SQLSTATEs for `SQLExecute()` include all those for `SQLExecDirect()` except for `HY009`, `HY090` and with the addition of the SQLSTATE in the table below. Any SQLSTATE that `SQLPrepare()` could return can also be returned on a call to `SQLExecute()` as a result of deferred prepare behavior.

Table 50. `SQLExecute` SQLSTATEs

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The specified <i>StatementHandle</i> was not in a prepared state. <code>SQLExecute()</code> was called without first calling <code>SQLPrepare()</code> .

### Authorization

None.

### Example

```
SQLHANDLE hstmt; /* statement handle */
SQLCHAR *stmt = (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? ";
SQLSMALLINT parameter1 = 0;

/* allocate a statement handle */
cliRC = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

/* ... */

/* prepare the statement */
cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);

/* ... */

/* bind parameter1 to the statement */
cliRC = SQLBindParameter(hstmt,
                        1,
                        SQL_PARAM_INPUT,
                        SQL_C_SHORT,
                        SQL_SMALLINT,
                        0,
                        0,
                        &parameter1,
                        0,
                        NULL);

/* ... */
parameter1 = 15;

/* execute the statement for parameter1 = 15 */
cliRC = SQLExecute(hstmt);
```

## SQLExtendedBind function (CLI) - Bind an array of columns

### Purpose

Specification:	CLI 6		
----------------	-------	--	--

SQLExtendedBind() is used to bind an array of columns or parameters instead of using repeated calls to SQLBindCol() or SQLBindParameter().

### Syntax

```
SQLRETURN      SQLExtendedBind (
                SQLHSTMT          StatementHandle, /* hstmt */
                SQLSMALLINT        fBindCol,
                SQLSMALLINT        cRecords,
                SQLSMALLINT *      pfCType,
                SQLPOINTER *       rgbValue,
                SQLINTEGER *       cbValueMax,
                SQLUINTEGER *      puiPrecisionCType,
                SQLSMALLINT *      psScaleCType,
                SQLINTEGER **      pcbValue,
                SQLINTEGER **      piIndicator,
                SQLSMALLINT *      pfParamType,
                SQLSMALLINT *      pfSQLType,
                SQLUINTEGER *      pcbColDef,
                SQLSMALLINT *      pibScale );
```

### Function arguments

Table 51. SQLExtendedBind() arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLSMALLINT	<i>fBindCol</i>	input	If SQL_TRUE then the result is similar to SQLBindCol(), otherwise, it is similar to SQLBindParameter().
SQLSMALLINT	<i>cRecords</i>	input	Number of columns or parameters to bind.
SQLSMALLINT *	<i>pfCType</i>	input	Array of values for the application data type.
SQLPOINTER *	<i>rgbValue</i>	input	Array of pointers to application data area.
SQLINTEGER *	<i>cbValueMax</i>	input	Array of maximum sizes for <i>rgbValue</i> .
SQLUINTEGER *	<i>puiPrecisionCType</i>	input	Array of decimal precision values. Each value is used only if the application data type of the corresponding record is SQL_C_DECIMAL_IBM.
SQLSMALLINT *	<i>psScaleCType</i>	input	Array of decimal scale values. Each value is used only if the application data type of the corresponding record is SQL_C_DECIMAL_IBM.
SQLINTEGER **	<i>pcbValue</i>	input	Array of pointers to length values.
SQLINTEGER **	<i>piIndicator</i>	input	Array of pointers to indicator values. The <i>piIndicator</i> argument allows the constants SQL_UNASSIGNED and SQL_DEFAULT_PARAM to pass through the method, when extended indicator feature is enabled using the SQL_ATTR_EXTENDED_INDICATORS attribute.

## SQLExtendedBind function (CLI) - Bind an array of columns

Table 51. SQLExtendedBind() arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	<i>pfParamType</i>	input	<p>Array of parameter types. Only used if <i>fBindCol</i> is FALSE.</p> <p>Each row in this array serves the same purpose as the SQLBindParameter() argument <i>InputOutputType</i>. It can be set to:</p> <ul style="list-style-type: none"> <li>• SQL_PARAM_INPUT</li> <li>• SQL_PARAM_INPUT_OUTPUT</li> <li>• SQL_PARAM_OUTPUT</li> </ul>
SQLSMALLINT *	<i>pfSQLType</i>	input	<p>Array of SQL data types. Only used if <i>fBindCol</i> is FALSE.</p> <p>Each row in this array serves the same purpose as the SQLBindParameter() argument <i>ParameterType</i>.</p>
SQLINTEGER *	<i>pcbColDef</i>	input	<p>Array of SQL precision values. Only used if <i>fBindCol</i> is FALSE.</p> <p>Each row in this array serves the same purpose as the SQLBindParameter() argument <i>ColumnSize</i>.</p>
SQLSMALLINT *	<i>pibScale</i>	input	<p>Array of SQL scale values. Only used if <i>fBindCol</i> is FALSE.</p> <p>Each row in this array serves the same purpose as the SQLBindParameter() argument <i>DecimalDigits</i>.</p>

### Usage

The argument *fBindCol* determines whether this function call is used to associate (bind):

- parameter markers in an SQL statement (as with SQLBindParameter()) - *fBindCol* = SQL\_FALSE
- columns in a result set (as with SQLBindCol()) - *fBindCol* = SQL\_TRUE

This function can be used to replace multiple calls to SQLBindCol() or SQLBindParameter(), however, important differences should be noted. Depending on how the *fBindCol* parameter has been set, the input expected by SQLExtendedBind() is similar to either SQLBindCol() or SQLBindParameter() with the following exceptions:

- When SQLExtendedBind() is set to SQLBindCol() mode:
  - *targetValuePtr* must be a positive integer that specifies in bytes, the maximum length of the data that will be in the returned column.
- When SQLExtendedBind() is set to SQLBindParameter() mode:
  - *ColumnSize* must be a positive integer that specifies the maximum length of the target column in bytes, where applicable.
  - *DecimalDigits* must be set to the correct scale for the target column, where applicable.
  - *ValueType* of SQL\_C\_DEFAULT should not be used.
  - If *ValueType* is a locator type, the corresponding *ParameterType* should be a matching locator type.
  - All *ValueType* to *ParameterType* mappings should be as closely matched as possible to minimize the conversion that CLI must perform.

## SQLExtendedBind function (CLI) - Bind an array of columns

Each array reference passed to `SQLExtendedBind()` must contain at least the number of elements indicated by `cRecords`. If the calling application fails to pass in sufficiently large arrays, CLI may attempt to read beyond the end of the arrays resulting in corrupt data or critical application failure.

Each array passed to `SQLExtendedBind()` is considered to be a deferred argument, which means the values in the array are examined and retrieved at the time of execution. As a result, ensure that each array is in a valid state and contains valid data when CLI executes using the values in the array. Following a successful execution, if a statement needs to be executed again, you do not need to call `SQLExtendedBind()` a second time if the handles passed to the original call to `SQLExtendedBind()` still refer to valid arrays.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 52. `SQLExtendedBind()` `SQLSTATEs`

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The conversion from the data value identified by a row in the <code>pfCType</code> argument to the data type identified by the <code>pfParamType</code> argument is not a meaningful conversion. (For example, conversion from <code>SQL_C_TYPE_DATE</code> to <code>SQL_DOUBLE</code> .)
07009	Invalid descriptor index	The value specified for the argument <code>cRecords</code> exceeded the maximum number of columns in the result set.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	A row in <code>pfParamType</code> or <code>pfSQLType</code> was not a valid data type or <code>SQL_C_DEFAULT</code> .
HY004	SQL data type out of range.	The value specified for the argument <code>pfParamType</code> is not a valid SQL data type.
HY009	Invalid argument value.	The argument <code>rgbValue</code> was a null pointer and the argument <code>cbValueMax</code> was a null pointer, and <code>pfParamType</code> is not <code>SQL_PARAM_OUTPUT</code> .
HY010	Function sequence error.	The function was called while in a data-at-execute ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.  The function was called while within a <code>BEGIN COMPOUND</code> and <code>END COMPOUND SQL</code> operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check was not consistent.



## SQLExtendedBind function (CLI) - Bind an array of columns

Table 52. SQLExtendedBind() SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The value specified for the argument <i>cbValueMax</i> is less than 1 and the argument the corresponding row in <i>pfParamType</i> or <i>pfSQLType</i> is either SQL_C_CHAR, SQL_C_BINARY or SQL_C_DEFAULT.
HY093	Invalid parameter number.	The value specified for a row in the argument <i>pfCType</i> was less than 1 or greater than the maximum number of parameters supported by the server.
HY094	Invalid scale value.	<p>The value specified for <i>pfParamType</i> was either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>DecimalDigits</i> was less than 0 or greater than the value for the argument <i>pcbColDef</i> (precision).</p> <p>The value specified for <i>pfParamType</i> was SQL_C_TYPE_TIMESTAMP and the value for <i>pfParamType</i> was either SQL_CHAR or SQL_VARCHAR and the value for <i>DecimalDigits</i> was less than 0 or greater than 9.</p> <p>The value specified for <i>pfParamType</i> was SQL_C_TIMESTAMP_EXT and the value for <i>DecimalDigits</i> was less than 0 or greater than 12.</p>
HY104	Invalid precision value.	The value specified for <i>pfParamType</i> was either SQL_DECIMAL or SQL_NUMERIC and the value specified by <i>pcbColDef</i> was less than 1.
HY105	Invalid parameter type.	<i>pfParamType</i> is not one of SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT.
HYC00	Driver not capable.	<p>CLI recognizes, but does not support the data type specified in the row in <i>pfParamType</i> or <i>pfSQLType</i>.</p> <p>A LOB locator C data type was specified, but the connected server does not support LOB data types.</p>

### Restrictions

None

## SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows)

### Deprecated

#### Note:

In ODBC 3.0, SQLExtendedFetch() has been deprecated and replaced with SQLFetchScroll().

Although this version of CLI continues to support SQLExtendedFetch(), use SQLFetchScroll() in your CLI programs so that they conform to the latest standards.

### Migrating to the new function

The statement:

## SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows)

```
SQLExtendedFetch(hstmt, SQL_FETCH_ABSOLUTE, 5, &rowCount, &rowStatus);
```

for example, would be rewritten using the new function as:

```
SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 5);
```

### Note:

The information returned in the *rowCount* and *rowStatus* parameters of SQLExtendedFetch() are handled by SQLFetchScroll() as follows:

- *rowCount*: SQLFetchScroll() returns the number of rows fetched in the buffer pointed to by the SQL\_ATTR\_ROWS\_FETCHED\_PTR statement attribute.
- *rowStatus*: SQLFetchScroll() returns the array of statuses for each row in the buffer pointed to by the SQL\_ATTR\_ROW\_STATUS\_PTR statement attribute.

---

## SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes

### Purpose

Specification:	CLI 6.0		
----------------	---------	--	--

SQLExtendedPrepare() is used to prepare a statement and set a group of statement attributes, all in one call.

This function can be used in place of a call to SQLPrepare() followed by a number of calls to SQLSetStmtAttr().

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLExtendedPrepareW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLExtendedPrepare(  
    SQLHSTMT      StatementHandle, /* hstmt */  
    SQLCHAR       *StatementText,  /* pszSqlStmt */  
    SQLINTEGER    TextLength,      /* cbSqlStmt */  
    SQLINTEGER    cParms,  
    SQLSMALLINT  sStmtType,  
    SQLINTEGER    cStmtAttrs,  
    SQLINTEGER    *piStmtAttr,  
    SQLINTEGER    *pvParams );
```

### Function arguments

Table 53. SQLExtendedPrepare() arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLCHAR *	<i>StatementText</i>	Input	SQL statement string.
SQLINTEGER	<i>TextLength</i>	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>StatementText</i> argument, or SQL_NTS if <i>StatementText</i> is null-terminated.
SQLINTEGER	<i>cParms</i>	Input	Number of parameter markers in statement.

## SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes

Table 53. SQLExtendedPrepare() arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>cStmtType</i>	Input	Statement type. For possible values see List of <i>cStmtType</i> Values.
SQLINTEGER	<i>cStmtAttrs</i>	Input	Number of statement attributes specified on this call.
SQLINTEGER *	<i>piStmtAttr</i>	Input	Array of statement attributes to set.
SQLINTEGER *	<i>pvParams</i>	Input	Array of corresponding statement attributes values to set.

### Usage

The first three arguments of this function are exactly the same as the arguments in SQLPrepare().

There are two requirements when using SQLExtendedPrepare():

1. The SQL statements will not be scanned for ODBC/vendor escape clauses. It behaves as if the SQL\_ATTR\_NOSCAN statement attribute is set to SQL\_NOSCAN. If the SQL statement contains ODBC/vendor escape clauses then SQLExtendedPrepare() cannot be used.
2. You must indicate in advance (through *cPars*) the number of parameter markers that are included in the SQL statement.

The *cPars* argument indicates the number of parameter markers in *StatementText*.

The argument *cStmtType* is used to indicate the type of statement that is being prepared. See List of *cStmtType* Values for the list of possible values.

The final three arguments are used to indicate a set of statement attributes to use. Set *cStmtAttrs* to the number of statement attributes specified on this call. Create two arrays, one to hold the list of statement attributes, one to hold the value for each. Use these arrays for *piStmtAttr* and *pvParams*.

### List of *cStmtType* Values

The argument *cStmtType* can be set to one of the following values:

- SQL\_CLI\_STMT\_UNDEFINED
- SQL\_CLI\_STMT\_ALTER\_TABLE
- SQL\_CLI\_STMT\_CREATE\_INDEX
- SQL\_CLI\_STMT\_CREATE\_TABLE
- SQL\_CLI\_STMT\_CREATE\_VIEW
- SQL\_CLI\_STMT\_DELETE\_SEARCHED
- SQL\_CLI\_STMT\_DELETE\_POSITIONED
- SQL\_CLI\_STMT\_GRANT
- SQL\_CLI\_STMT\_INSERT
- SQL\_CLI\_STMT\_INSERT\_VALUES
- SQL\_CLI\_STMT\_REVOKE
- SQL\_CLI\_STMT\_SELECT
- SQL\_CLI\_STMT\_UPDATE\_SEARCHED
- SQL\_CLI\_STMT\_UPDATE\_POSITIONED
- SQL\_CLI\_STMT\_CALL
- SQL\_CLI\_STMT\_SELECT\_FOR\_UPDATE
- SQL\_CLI\_STMT\_WITH
- SQL\_CLI\_STMT\_SELECT\_FOR\_FETCH

## SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes

- SQL\_CLI\_STMT\_VALUES
- SQL\_CLI\_STMT\_CREATE\_TRIGGER
- SQL\_CLI\_STMT\_SELECT\_OPTIMIZE\_FOR\_NROWS
- SQL\_CLI\_STMT\_SELECT\_INTO
- SQL\_CLI\_STMT\_CREATE\_PROCEDURE
- SQL\_CLI\_STMT\_CREATE\_FUNCTION
- SQL\_CLI\_STMT\_SET\_CURRENT\_QUERY\_OPT

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 54. SQLExtendedPrepare SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01504	The UPDATE or DELETE statement does not include a WHERE clause.	<i>StatementText</i> contained an UPDATE or DELETE statement which did not contain a WHERE clause.
01508	Statement disqualified for blocking.	The statement was disqualified for blocking for reasons other than storage.
01S02	Option value changed.	CLI did not support a value specified in <i>*pvParams</i> , or a value specified in <i>*pvParams</i> was invalid because of SQL constraints or requirements, so CLI substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
21S01	Insert value list does not match column list.	<i>StatementText</i> contained an INSERT statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degrees of derived table does not match column list.	<i>StatementText</i> contained a CREATE VIEW statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22018	Invalid character value for cast specification.	<i>*StatementText</i> contained an SQL statement that contained a literal or parameter and the value was incompatible with the data type of the associated table column.
22019	Invalid escape character	The argument <i>StatementText</i> contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	The argument <i>StatementText</i> contained "LIKE <i>pattern value</i> ESCAPE <i>escape character</i> " in the WHERE clause, and the character following the escape character in the pattern value was not one of "%" or "_".
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
34000	Invalid cursor name.	<i>StatementText</i> contained a positioned DELETE or a positioned UPDATE and the cursor referenced by the statement being executed was not open.

## SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes

Table 54. SQLExtendedPrepare SQLSTATES (continued)

SQLSTATE	Description	Explanation
37xxx <sup>a</sup>	Invalid SQL syntax.	<i>StatementText</i> contained one or more of the following: <ul style="list-style-type: none"> <li>an SQL statement that the connected database server could not prepare</li> <li>a statement containing a syntax error</li> </ul>
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to deadlock or timeout.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42xxx <sup>a</sup>	Syntax Error or Access Rule Violation.	<b>425xx</b> indicates the authorization ID does not have permission to execute the SQL statement contained in <i>StatementText</i> .  Other <b>42xxx</b> SQLSTATES indicate a variety of syntax or access problems with the statement.
58004	Unexpected system failure.	Unrecoverable system error.
S0001	Database object already exists.	<i>StatementText</i> contained a CREATE TABLE or CREATE VIEW statement and the table name or view name specified already existed.
S0002	Database object does not exist.	<i>StatementText</i> contained an SQL statement that references a table name or a view name which did not exist.
S0011	Index already exists.	<i>StatementText</i> contained a CREATE INDEX statement and the specified index name already existed.
S0012	Index not found.	<i>StatementText</i> contained a DROP INDEX statement and the specified index name did not exist.
S0021	Column already exists.	<i>StatementText</i> contained an ALTER TABLE statement and the column specified in the ADD clause was not unique or identified an existing column in the base table.
S0022	Column not found.	<i>StatementText</i> contained an SQL statement that references a column name which did not exist.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	<i>StatementText</i> was a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY011	Operation invalid at this time.	The <i>Attribute</i> was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS and the statement was prepared.

## SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes

Table 54. SQLExtendedPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY017	Invalid use of an automatically allocated descriptor handle.	The <i>Attribute</i> argument was SQL_ATTR_IMP_ROW_DESC or SQL_ATTR_IMP_PARAM_DESC. The <i>Attribute</i> argument was SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and the value in <i>*ValuePtr</i> was an implicitly allocated descriptor handle.
HY024	Invalid attribute value.	Given the specified <i>Attribute</i> value, an invalid value was specified in <i>*ValuePtr</i> . (CLI returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE. For all other connection and statement attributes, the driver must verify the value specified in <i>*ValuePtr</i> .)
HY090	Invalid string or buffer length.	The argument <i>TextLength</i> was less than 1, but not equal to SQL_NTS.
HY092	Option type out of range.	The value specified for the argument <i>Attribute</i> was not valid for this version of CLI.
HYC00	Driver not capable.	The value specified for the argument <i>Attribute</i> was a valid connection or statement attribute for the version of the CLI driver, but was not supported by the data source.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Note:

**a** xxx refers to any SQLSTATE with that class code. Example, 37xxx refers to any SQLSTATE in the 37 class.

**Note:** Not all DBMSs report all of the above diagnostic messages at prepare time. If deferred prepare is left on as the default behavior (controlled by the SQL\_ATTR\_DEFERRED\_PREPARE statement attribute), then these errors could occur when the PREPARE is flowed to the server. The application must be able to handle these conditions when calling functions that cause this flow. These functions include SQLExecute(), SQLDescribeParam(), SQLNumResultCols(), SQLDescribeCol(), and SQLColAttribute().

### Restrictions

When accessing IDS data servers, only IDS data server specific SQLExtendedPrepare() attributes are supported. If any SQLExtendedPrepare() attributes not supported by the IDS data server are used, a "Driver not capable" error is returned.

## SQLExtendedProcedures function (CLI) - Get list of procedure names

The SQLExtendedProcedures() function returns a list of stored procedure names that are registered at the server, and which match the specified search pattern.

The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

## SQLExtendedProcedures function (CLI) - Get list of procedure names

### Purpose

Specification:	CLI 9.7		
----------------	---------	--	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLExtendedProceduresW()`. For information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

Modules are an extension to the concept of schemas. Applications that connect to DB2 version 9.7 or later data servers can create modules inside their schema and can create procedures inside the modules. The fully qualified name of a procedure in a module would be <SCHEMA NAME>.<MODULE NAME>.<PROCEDURE NAME>. The `SQLExtendedProcedures()` and `SQLExtendedProcedureColumns()` functions provide information about modules. These functions are not part of the current ODBC specification. For more information, see “Modules” in *SQL Procedural Languages: Application Enablement and Support*.

### Syntax

```
SQLRETURN SQLExtendedProcedures (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *CatalogName, /* szProcCatalog */
    SQLSMALLINT NameLength1, /* cbProcCatalog */
    SQLCHAR *SchemaName, /* szProcSchema */
    SQLSMALLINT NameLength2, /* cbProcSchema */
    SQLCHAR *ProcName, /* szProcName */
    SQLSMALLINT NameLength3, /* cbProcName */
    SQLCHAR *ProcModule, /* szProcModule */
    SQLSMALLINT NameLength4; /* cbProcModule */
)
```

### Function arguments

Table 55. *SQLExtendedProcedures* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and <code>SQL_SUCCESS</code> is returned. Otherwise, this is a valid filter for DBMSs that supports 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or <code>SQL_NTS</code> if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by schema name.  For DB2 for MVS/ESA V 4.1 and later, all the stored procedures are in one schema; the only acceptable value for the <i>SchemaName</i> argument is a null pointer. If a value is specified, an empty result set and <code>SQL_SUCCESS</code> are returned. For DB2 Database for Linux, UNIX, and Windows, <i>SchemaName</i> can contain a valid pattern value. For more information about valid search patterns, see the catalog functions input arguments.

## SQLExtendedProcedures function (CLI) - Get list of procedure names

Table 55. SQLExtendedProcedures arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>ProcName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ProcName</i> , or SQL_NTS if <i>ProcName</i> is null-terminated.
SQLCHAR *	<i>ProcModule</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by module name.
SQLSMALLINT	<i>NameLength4</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ProcModule</i> , or SQL_NTS if <i>ProcModule</i> is null-terminated.

### Usage

The result set that is returned by the SQLExtendedProcedures() function contains the columns that are listed in Columns returned by SQLExtendedProcedures in the order given. The rows are ordered by PROCEDURE\_CAT, PROCEDURE\_SCHEMA, and PROCEDURE\_NAME.

In many cases, calls to the SQLExtendedProcedures() function map to a complex and thus expensive query against the system catalog, so you should use the calls sparingly, and save the results rather than repeating calls.

Call SQLGetInfo() with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

If the SQL\_ATTR\_LONGDATA\_COMPAT connection attribute is set, LOB column types are reported as LONG VARCHAR, LONG VARBINARY, or LONG VARGRAPHIC types.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

If the stored procedure is at a DB2 for MVS/ESA V4.1 up to V6 server, the name of the stored procedures must be registered in the server's SYSIBM.SYSPROCEDURES catalog table. For V8 and later servers, the stored procedure must be registered in the server's SYSIBM.SYSROUTINES and SYSIBM.SYSPARAMS catalog tables.

For other versions of DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set is returned.

In Version 9.7 Fix Pack 5, you can specify \*ALL as a value in the *SchemaName* to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not



## SQLExtendedProcedures function (CLI) - Get list of procedure names

required to specify \*ALL, as this behavior is the default in CLI. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL.

### Columns returned by SQLExtendedProcedures

**Column 1 PROCEDURE\_CAT (VARCHAR(128))**

The procedure catalog name. The value is NULL if this procedure does not have catalogs.

**Column 2 PROCEDURE\_SCHEM (VARCHAR(128))**

The name of the schema that contains PROCEDURE\_NAME.

**Column 3 PROCEDURE\_NAME (VARCHAR(128) NOT NULL)**

The name of the procedure.

**Column 4 NUM\_INPUT\_PARAMS (INTEGER not NULL)**

The number of input parameters. INOUT parameters are not counted as part of this number.

To determine information regarding INOUT parameters, examine the COLUMN\_TYPE column that is returned by SQLProcedureColumns().

**Column 5 NUM\_OUTPUT\_PARAMS (INTEGER not NULL)**

The number of output parameters. INOUT parameters are not counted as part of this number.

To determine information regarding INOUT parameters, examine the COLUMN\_TYPE column that is returned by SQLProcedureColumns().

**Column 6 NUM\_RESULT\_SETS (INTEGER not NULL)**

The number of result sets that are returned by the procedure.

You should not use this column, it is reserved for future use by ODBC.

**Column 7 REMARKS (VARCHAR(254))**

Contains the descriptive information about the procedure.

**Column 8 PROCEDURE\_TYPE (SMALLINT)**

Defines the procedure type:

- SQL\_PT\_UNKNOWN: It cannot be determined whether the procedure returns a value.
- SQL\_PT\_PROCEDURE: The returned object is a procedure that does not have a return value
- SQL\_PT\_FUNCTION: The returned object is a function that has a return value.

CLI always returns SQL\_PT\_PROCEDURE.

**Column 9 SPECIFIC\_NAME (VARCHAR(128))**

The unique specific name of PROCEDURE\_NAME.

**Column 10 PROCEDURE\_MODULE (VARCHAR(128))**

The name of the module that contains PROCEDURE\_NAME within the schema.

**Note:**

- The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the SQLExtendedProcedures() result set in ODBC.
- If two modules contain procedures that share the same name, the SQLExtendedProcedures() function returns details about both procedures.

## SQLExtendedProcedures function (CLI) - Get list of procedure names

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 56. SQLExtendedProcedures SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (<code>SQLParamData()</code>, <code>SQLPutData()</code>) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name-length arguments was less than 0, but not equal to <code>SQL_NTS</code> .
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

### Restrictions

If an application is connected to a DB2 server that does not provide support for a stored procedure catalog, or does not provide support for stored procedures, the `SQLExtendedProcedures()` function returns an empty result set.

The `SQLExtendedProcedures()` function is currently supported only with DB2 version 9.7 or later data servers.

## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

The SQLExtendedProcedureColumns() function returns a list of input and output parameters that are associated with a stored procedure. The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

### Purpose

Specification:	CLI 9.7		
----------------	---------	--	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is SQLExtendedProcedureColumnsW(). For information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

Modules are an extension to the concept of schemas. Applications that connect to DB2 version 9.7 or later data servers can create modules inside their schema, and can create procedures inside the modules. The fully qualified name of a procedure in a module would be <SCHEMA NAME>.<MODULE NAME>.<PROCEDURE NAME>. The SQLExtendedProcedures() and SQLExtendedProcedureColumns() functions provide information about modules. These functions are not part of the current ODBC specification. See in *SQL Procedural Languages: Application Enablement and Support* for more information.

### Syntax

```
SQLRETURN SQLExtendedProcedureColumns (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLCHAR           *CatalogName,    /* szProcCatalog */
    SQLSMALLINT       NameLength1,     /* cbProcCatalog */
    SQLCHAR           *SchemaName,     /* szProcSchema */
    SQLSMALLINT       NameLength2,     /* cbProcSchema */
    SQLCHAR           *ProcName,       /* szProcName */
    SQLSMALLINT       NameLength3,     /* cbProcName */
    SQLCHAR           *ColumnName,     /* szColumnName */
    SQLSMALLINT       NameLength4),    /* cbColumnName */
    SQLCHAR           *ProcModule,     /* szProcModule */
    SQLSMALLINT       NameLength5;    /* cbProcModule */
```

### Function arguments

Table 57. SQLExtendedProcedureColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	Input	A statement handle.
SQLCHAR *	CatalogName	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and CatalogName is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS is returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	NameLength1	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store CatalogName, or SQL_NTS if CatalogName is null-terminated.

## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

Table 57. SQLExtendedProcedureColumns arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by schema name.  For DB2 Database for Linux, UNIX, and Windows, <i>SchemaName</i> can contain a valid pattern value. For more information about valid search patterns, see the catalog functions input arguments.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>ProcName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by procedure name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ProcName</i> , or SQL_NTS if <i>ProcName</i> is null-terminated.
SQLCHAR *	<i>ColumnName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by parameter name. Use this argument to further qualify the result set that is already restricted by specifying a non-empty value for <i>ProcName</i> , <i>SchemaName</i> , or both.
SQLSMALLINT	<i>NameLength4</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ColumnName</i> , or SQL_NTS if <i>ColumnName</i> is null-terminated.
SQLCHAR *	<i>ProcModule</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by parameter name. Use this argument to further qualify the result set that is already restricted by specifying a non-empty value for <i>ProcName</i> , <i>SchemaName</i> , or <i>ColumnName</i> .
SQLSMALLINT	<i>NameLength5</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ProcModule</i> , or SQL_NTS if <i>ProcModule</i> is null-terminated.

### Usage

The SQLExtendedProcedureColumns() function returns the information in a result set, that is ordered by PROCEDURE\_CAT, PROCEDURE\_SCHEM, PROCEDURE\_NAME, COLUMN\_TYPE and PROCEDURE\_MODULE. Columns returned by SQLExtendedProcedureColumns lists the columns in the result set. Columns that are beyond the last column might be defined in future releases.

In many cases, calls to the SQLExtendedProcedureColumns() function map to a complex and thus expensive query against the system catalog, therefore you should use these calls sparingly, and save the results rather than repeating calls.

Call the SQLGetInfo() function with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, and COLUMN\_NAME columns that are supported by the connected DBMS.

## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

If the SQL\_ATTR\_LONGDATA\_COMPAT connection attribute is set, LOB column types are reported as LONG VARCHAR, LONG VARBINARY or LONG VARGRAPHIC types.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

For versions of other DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set is returned.

CLI returns information about the input, input/output, and output parameters that are associated with the stored procedure, but cannot return descriptor information for any result sets that the stored procedure might return.

In Version 9.7 Fix Pack 5, you can specify \*ALL as a value in the *SchemaName* to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL.

### Columns returned by SQLExtendedProcedureColumns

#### Column 1 PROCEDURE\_CAT (VARCHAR(128))

The name of the procedure catalog. The value is NULL if this procedure does not have catalogs.

#### Column 2 PROCEDURE\_SCHEM (VARCHAR(128))

The name of the schema containing PROCEDURE\_NAME.

#### Column 3 PROCEDURE\_NAME (VARCHAR(128))

The name of the procedure.

#### Column 4 COLUMN\_NAME (VARCHAR(128))

The name of the parameter.

#### Column 5 COLUMN\_TYPE (SMALLINT not NULL)

Identifies the type of information that is associated with this row. The values can be:

- SQL\_PARAM\_TYPE\_UNKNOWN : The parameter type is unknown.

**Note:** This is not returned.

- SQL\_PARAM\_INPUT: This parameter is an input parameter.
- SQL\_PARAM\_INPUT\_OUTPUT: This parameter is an input / output parameter.
- SQL\_PARAM\_OUTPUT: This parameter is an output parameter.
- SQL\_RETURN\_VALUE: The procedure column is the return value of the procedure.

**Note:** This is not returned.

- SQL\_RESULT\_COL: This parameter is actually a column in the result set.

**Note:** This is not returned.

#### Column 6 DATA\_TYPE (SMALLINT not NULL)

An SQL data type.

## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

### Column 7 TYPE\_NAME (VARCHAR(128) not NULL)

A character string that represents the name of the data type that corresponds to DATA\_TYPE.

### Column 8 COLUMN\_SIZE (INTEGER)

For XML arguments in SQL routines, zero is returned (as XML arguments have no length). For cataloged external routines, however, XML parameters are declared as XML AS CLOB(n), in which case COLUMN\_SIZE is the cataloged length, n.

If the DATA\_TYPE column value denotes a character or binary string, this column contains the maximum length in SQLCHAR or SQLWCHAR elements. If the DATA\_TYPE column value is a graphic (DBCS) string, COLUMN\_SIZE is the number of double byte SQLCHAR or SQLWCHAR elements for the parameter.

For date, time, and timestamp data types, COLUMN\_SIZE is the total number of SQLCHAR or SQLWCHAR elements that are required to display the value when converted to character data type.

For numeric data types, this is either the total number of digits, or the total number of bits that are allowed in the column, depending on the value in the NUM\_PREC\_RADIX column in the result set.

See the table of data type precision.

### Column 9 BUFFER\_LENGTH (INTEGER)

The maximum number of bytes for the associated C buffer to store data from this parameter if SQL\_C\_DEFAULT is specified on the SQLBindCol(), SQLGetData() and SQLBindParameter() calls. This length excludes any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

For XML arguments in SQL routines, zero is returned (as XML arguments have no length). For cataloged external routines, however, XML parameters are declared as XML AS CLOB(n), in which case BUFFER\_LENGTH is the cataloged length, n.

See the table of data type length.

### Column 10 DECIMAL\_DIGITS (SMALLINT)

The scale of the parameter. NULL is returned for data types where scale is not applicable.

See the table of data type scale.

### Column 11 NUM\_PREC\_RADIX (SMALLINT)

Either 10, 2, or NULL. If DATA\_TYPE is an approximate numeric data type, this column contains the value 2, and the COLUMN\_SIZE column contains the number of bits that are allowed in the parameter.

If DATA\_TYPE is an exact numeric data type, this column contains the value 10, and the COLUMN\_SIZE and DECIMAL\_DIGITS columns contain the number of decimal digits that are allowed for the parameter.

For numeric data types, the DBMS can return a NUM\_PREC\_RADIX of either 10 or 2.

NULL is returned for data types where radix is not applicable.

### Column 12 NULLABLE (SMALLINT not NULL)

SQL\_NO\_NULLS if the parameter does not accept NULL values.

SQL\_NULLABLE if the parameter accepts NULL values.

## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

### Column 13 REMARKS (VARCHAR(254))

Might contain descriptive information about the parameter.

### Column 14 COLUMN\_DEF (VARCHAR)

The default value of the column.

If NULL is specified as the default value, this column is the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, this column contains TRUNCATED, with no enclosing single quotation marks. If no default value is specified, this column is NULL.

You can use the value of COLUMN\_DEF to generate a new column definition, except when it contains the value TRUNCATED.

### Column 15 SQL\_DATA\_TYPE (SMALLINT not NULL)

The value of the SQL data type as it is displayed in the SQL\_DESC\_TYPE field of the descriptor. This column is the same as the DATA\_TYPE column except for datetime data types (CLI does not support interval data types).

For datetime data types, the SQL\_DATA\_TYPE field in the result set is SQL\_DATETIME, and the SQL\_DATETIME\_SUB field returns the subcode for the specific datetime data type (SQL\_CODE\_DATE, SQL\_CODE\_TIME or SQL\_CODE\_TIMESTAMP).

### Column 16 SQL\_DATETIME\_SUB (SMALLINT)

The subtype code for datetime data types. For all other data types this column returns a NULL (including interval data types, which CLI does not support).

### Column 17 CHAR\_OCTET\_LENGTH (INTEGER)

The maximum length in bytes of a character data type column. For all other data types, this column returns a NULL.

### Column 18 ORDINAL\_POSITION (INTEGER NOT NULL)

Contains the ordinal position of the parameter that is given by COLUMN\_NAME in this result set. The ORDINAL\_POSITION is the ordinal position of the argument to be provided on the CALL statement. The leftmost argument has an ordinal position of 1.

### Column 19 IS\_NULLABLE (Varchar)

- "NO" if the column does not include NULLs.
- "YES" if the column can include NULLs.
- Zero-length string if the nullability is unknown.

ISO rules are followed to determine nullability.

An ISO SQL-compliant DBMS cannot return an empty string.

The value that is returned for this column is different than the value that is returned for the NULLABLE column. See the description of the NULLABLE column.

### Column 20 SPECIFIC\_NAME (VARCHAR(128))

The unique specific name of PROCEDURE\_NAME.

### Column 21 PROCEDURE\_MODULE (VARCHAR(128))

The name of the module containing PROCEDURE\_NAME within the schema.

**Note:**

## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

- The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the SQLExtendedProcedureColumns() result set in ODBC.
- If two modules contain procedures that share the same name, the SQLExtendedProcedureColumns() function returns details about both procedures.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 58. SQLExtendedProcedureColumns SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor is already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42601	PARMLIST syntax error.	The PARMLIST value that is in the stored procedures catalog table contains a syntax error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().



## SQLExtendedProcedureColumns function (CLI) - Get input/output parameter information for a procedure

### Restrictions

SQLExtendedProcedureColumns() does not return information about the attributes of result sets that might be returned from stored procedures.

If an application is connected to a DB2 server that does not provide support for a stored procedure catalog, or does not provide support for stored procedures, SQLExtendedProcedureColumns() will return an empty result set.

SQLExtendedProcedureColumns() is currently only supported with DB2 Version 9.7 or later.

### Example

```
/* get input/output parameter information for a procedure including
extended information */
cliRC = SQLExtendedProcedureColumns(hstmt,
    "CatalogName",
    SQL_NTS,
    "SchemaName",
    SQL_NTS,
    "ProcName",
    SQL_NTS,
    "ColumnName",
    SQL_NTS,
    "ModuleName",
    SQL_NTS );
```

---

## SQLFetch function (CLI) - Fetch next row

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLFetch() advances the cursor to the next row of the result set, and retrieves any bound columns.

Columns can be bound to:

- application storage
- LOB locators
- LOB file references

When SQLFetch() is called, the appropriate data transfer is performed, along with any data conversion if conversion was indicated when the column was bound. The columns can also be received individually after the fetch, by calling SQLGetData().

SQLFetch() can only be called after a result set has been generated (using the same statement handle) by either executing a query, calling SQLGetTypeInfo() or calling a catalog function.

### Syntax

```
SQLRETURN SQLFetch (SQLHSTMT StatementHandle); /* hstmt */
```

## SQLFetch function (CLI) - Fetch next row

### Function arguments

Table 59. SQLFetch arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle

### Usage

SQLFetch() can only be called after a result set has been generated on the same statement handle. Before SQLFetch() is called the first time, the cursor is positioned before the start of the result set.

The number of application variables bound with SQLBindCol() must not exceed the number of columns in the result set or SQLFetch() will fail.

If SQLBindCol() has not been called to bind any columns, then SQLFetch() does not return data to the application, but just advances the cursor. In this case SQLGetData() could be called to obtain all of the columns individually. If the cursor is a multirow cursor (that is, the SQL\_ATTR\_ROW\_ARRAY\_SIZE is greater than 1), SQLGetData() can be called only if SQL\_GD\_BLOCK is returned when SQLGetInfo() is called with an *InfoType* of SQL\_GETDATA\_EXTENSIONS. (Not all DB2 data sources support SQL\_GD\_BLOCK.) Data in unbound columns is discarded when SQLFetch() advances the cursor to the next row. For fixed length data types, or small variable length data types, binding columns provides better performance than using SQLGetData().

If LOB values are too large to be retrieved in one fetch, they can be retrieved in pieces by either using SQLGetData() (which can be used for any column type), or by binding a LOB locator, and using SQLGetSubString().

If any bound storage buffer is not large enough to hold the data returned by SQLFetch(), the data will be truncated. If character data is truncated, SQL\_SUCCESS\_WITH\_INFO is returned, and an SQLSTATE is generated indicating truncation. The SQLBindCol() deferred output argument *pcbValue* will contain the actual length of the column data retrieved from the server. The application should compare the actual output length to the input buffer length (*pcbValue* and *cbValueMax* arguments from SQLBindCol()) to determine which character columns have been truncated.

Truncation of numeric data types is reported as a warning if the truncation involves digits to the right of the decimal point. If truncation occurs to the left of the decimal point, an error is returned (refer to the diagnostics section).

Truncation of graphic data types is treated the same as character data types, except that the *rgbValue* buffer is filled to the nearest multiple of two bytes that is still less than or equal to the *cbValueMax* specified in SQLBindCol(). Graphic (DBCS) data transferred between CLI and the application is not null-terminated if the C buffer type is SQL\_C\_CHAR (unless the CLI/ODBC configuration keyword PATCH1 includes the value 64). If the buffer type is SQL\_C\_DBCHAR, then null-termination of graphic data does occur.

Truncation is also affected by the SQL\_ATTR\_MAX\_LENGTH statement attribute. The application can specify that CLI should not report truncation by calling SQLSetStmtAttr() with SQL\_ATTR\_MAX\_LENGTH and a value for the maximum length to return for any one column, and by allocating a *rgbValue* buffer of the

## SQLFetch function (CLI) - Fetch next row

same size (plus the null-terminator). If the column data is larger than the set maximum length, `SQL_SUCCESS` will be returned and the maximum length, not the actual length will be returned in *pcbValue*.

When all the rows have been retrieved from the result set, or the remaining rows are not needed, `SQLCloseCursor()` or `SQLFreeStmt()` with an option of `SQL_CLOSE` or `SQL_DROP` should be called to close the cursor and discard the remaining data and associated resources.

An application cannot mix `SQLFetch()` with `SQLExtendedFetch()` calls on the same statement handle. It can, however, mix `SQLFetch()` with `SQLFetchScroll()` calls on the same statement handle. Note that `SQLExtendedFetch()` has been deprecated and replaced with `SQLFetchScroll()`.

### Positioning the cursor

When the result set is created, the cursor is positioned before the start of the result set. `SQLFetch()` fetches the next rowset. It is equivalent to calling `SQLFetchScroll()` with *FetchOrientation* set to `SQL_FETCH_NEXT`.

The `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute specifies the number of rows in the rowset. If the rowset being fetched by `SQLFetch()` overlaps the end of the result set, `SQLFetch()` returns a partial rowset. That is, if  $S + R - 1$  is greater than  $L$ , where  $S$  is the starting row of the rowset being fetched,  $R$  is the rowset size, and  $L$  is the last row in the result set, then only the first  $L - S + 1$  rows of the rowset are valid. The remaining rows are empty and have a status of `SQL_ROW_NOROW`.

Refer to the cursor positioning rules of `SQL_FETCH_NEXT` for `SQLFetchScroll()` for more information.

After `SQLFetch()` returns, the current row is the first row of the rowset.

### Row status array

`SQLFetch()` sets values in the row status array in the same manner as `SQLFetchScroll()` and `SQLBulkOperations()`. The row status array is used to return the status of each row in the rowset. The address of this array is specified with the `SQL_ATTR_ROW_STATUS_PTR` statement attribute.

### Rows fetched buffer

`SQLFetch()` returns the number of rows fetched in the rows fetched buffer including those rows for which no data was returned. The address of this buffer is specified with the `SQL_ATTR_ROWSFETCHED_PTR` statement attribute. The buffer is set by `SQLFetch()` and `SQLFetchScroll()`.

### Error handling

Errors and warnings can apply to individual rows or to the entire function. They can be retrieved using the `SQLGetDiagField()` function.

Errors and Warnings on the Entire Function

If an error applies to the entire function, such as `SQLSTATE HYT00` (Timeout expired) or `SQLSTATE 24000` (Invalid cursor state), `SQLFetch()` returns

## SQLFetch function (CLI) - Fetch next row

SQL\_ERROR and the applicable SQLSTATE. The contents of the rowset buffers are undefined and the cursor position is unchanged.

If a warning applies to the entire function, SQLFetch() returns SQL\_SUCCESS\_WITH\_INFO and the applicable SQLSTATE. The status records for warnings that apply to the entire function are returned before the status records that apply to individual rows.

Errors and warnings in individual rows

If an error (such as SQLSTATE 22012 (Division by zero)) or a warning (such as SQLSTATE 01004 (Data truncated)) applies to a single row, SQLFetch() returns SQL\_SUCCESS\_WITH\_INFO, unless an error occurs in every row, in which case SQL\_ERROR is returned. SQLFetch() also:

- Sets the corresponding element of the row status array to SQL\_ROW\_ERROR for errors or SQL\_ROW\_SUCCESS\_WITH\_INFO for warnings.
- Adds zero or more status records containing SQLSTATES for the error or warning.
- Sets the row and column number fields in the status records. If SQLFetch() cannot determine a row or column number, it sets that number to SQL\_ROW\_NUMBER\_UNKNOWN or SQL\_COLUMN\_NUMBER\_UNKNOWN respectively. If the status record does not apply to a particular column, SQLFetch() sets the column number to SQL\_NO\_COLUMN\_NUMBER.

SQLFetch() returns the status records in row number order. That is, it returns all status records for unknown rows (if any), then all status records for the first row (if any), then all status records for the second row (if any), and so on. The status records for each individual row are ordered according to the normal rules for ordering status records, described in SQLGetDiagField().

### Descriptors and SQLFetch

The following sections describe how SQLFetch() interacts with descriptors.

Argument mappings

The driver does not set any descriptor fields based on the arguments of SQLFetch().

Other descriptor fields

The following descriptor fields are used by SQLFetch():

Table 60. Descriptor fields

Descriptor field	Desc.	Location	Set through
SQL_DESC_ARRAY_SIZE	ARD	header	SQL_ATTR_ROW_ARRAY_SIZE statement attribute
SQL_DESC_ARRAY_STATUS_PTR	IRD	header	SQL_ATTR_ROW_STATUS_PTR statement attribute
SQL_DESC_BIND_OFFSET_PTR	ARD	header	SQL_ATTR_ROW_BIND_OFFSET_PTR statement attribute
SQL_DESC_BIND_TYPE	ARD	header	SQL_ATTR_ROW_BIND_TYPE statement attribute
SQL_DESC_COUNT	ARD	header	ColumnNumber argument of SQLBindCol()

Table 60. Descriptor fields (continued)

Descriptor field	Desc.	Location	Set through
SQL_DESC_DATA_PTR	ARD	records	<i>TargetValuePtr</i> argument of <code>SQLBindCol()</code>
SQL_DESC_INDICATOR_PTR	ARD	records	<i>StrLen_or_IndPtr</i> argument in <code>SQLBindCol()</code>
SQL_DESC_OCTET_LENGTH	ARD	records	<i>BufferLength</i> argument in <code>SQLBindCol()</code>
SQL_DESC_OCTET_LENGTH_PTR	ARD	records	<i>StrLen_or_IndPtr</i> argument in <code>SQLBindCol()</code>
SQL_DESC_ROWS_PROCESSED_PTR	IRD	header	<code>SQL_ATTR_ROWS_FETCHED_PTR</code> statement attribute
SQL_DESC_TYPE	ARD	records	<i>TargetType</i> argument in <code>SQLBindCol()</code>

All descriptor fields can also be set through `SQLSetDescField()`.

#### Separate length and indicator buffers

Applications can bind a single buffer or two separate buffers to be used to hold length and indicator values. When an application calls `SQLBindCol()`, `SQL_DESC_OCTET_LENGTH_PTR` and `SQL_DESC_INDICATOR_PTR` fields of the ARD are set to the same address, which is passed in the *StrLen\_or\_IndPtr* argument. When an application calls `SQLSetDescField()` or `SQLSetDescRec()`, it can set these two fields to different addresses.

`SQLFetch()` determines whether the application has specified separate length and indicator buffers. In this case, when the data is not NULL, `SQLFetch()` sets the indicator buffer to 0 and returns the length in the length buffer. When the data is NULL, `SQLFetch()` sets the indicator buffer to `SQL_NULL_DATA` and does not modify the length buffer.

#### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_STILL_EXECUTING`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

`SQL_NO_DATA_FOUND` is returned if there are no rows in the result set, or previous `SQLFetch()` calls have fetched all the rows from the result set.

If all the rows have been fetched, the cursor is positioned after the end of the result set.

#### Diagnostics

Table 61. SQLFetch SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data returned for one or more columns was truncated. String values or numeric values are right truncated. ( <code>SQL_SUCCESS_WITH_INFO</code> is returned if no error occurred.)
07002	Too many columns.	A column number specified in the binding for one or more columns was greater than the number of columns in the result set.
07006	Invalid conversion.	The data value could not be converted in a meaningful manner to the data type specified by <i>fCType</i> in <code>SQLBindCol()</code>

## SQLFetch function (CLI) - Fetch next row

Table 61. SQLFetch SQLSTATES (continued)

SQLSTATE	Description	Explanation
07009	Invalid descriptor index	Column 0 was bound but bookmarks are not being used (the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF).
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument <i>pcbValue</i> in SQLBindCol() was a null pointer and the value of the corresponding column is null. There is no means to report SQL_NULL_DATA. The pointer specified for the argument <i>IndicatorValue</i> in SQLBindFileToCol() was a null pointer and the value of the corresponding LOB column is NULL. There is no means to report SQL_NULL_DATA.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for one or more columns would have caused the whole part of the number to be truncated either at the time of assignment or in computing an intermediate result.  A value from an arithmetic expression was returned which resulted in division by zero. <b>Note:</b> The associated cursor is undefined if this error is detected by DB2 Database for Linux, UNIX, and Windows. If the error was detected by CLI or by other IBM RDBMSs, the cursor will remain open and continue to advance on subsequent fetch calls.
22005	Error in assignment.	A returned value was incompatible with the data type of binding.  A returned LOB locator was incompatible with the data type of the bound column.
22007	Invalid datetime format.	Conversion from character a string to a datetime format was indicated, but an invalid string representation or value was specified, or the value was an invalid date.  The value of a date, time, or timestamp does not conform to the syntax for the specified data type.
22008	Datetime field overflow.	Datetime field overflow occurred; for example, an arithmetic operation on a date or timestamp has a result that is not within the valid range of dates, or a datetime value cannot be assigned to a bound variable because it is too small.
22012	Division by zero is invalid.	A value from an arithmetic expression was returned which resulted in division by zero.
24000	Invalid cursor state.	The previous SQL statement executed on the statement handle was not a query.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.

Table 61. SQLFetch SQLSTATEs (continued)

SQLSTATE	Description	Explanation
428A1	Unable to access a file referenced by a host file variable.	<p>This can be raised for any of the following scenarios. The associated reason code in the text identifies the particular error:</p> <ul style="list-style-type: none"> <li>• 01 - The file name length is invalid, or the file name, the path or both has an invalid format.</li> <li>• 02 - The file option is invalid. It must have one of the following values: <ul style="list-style-type: none"> <li>SQL_FILE_READ -read from an existing file</li> <li>SQL_FILE_CREATE -create a new file for write</li> <li>SQL_FILE_OVERWRITE -overwrite an existing file. If the file does not exist, create the file.</li> <li>SQL_FILE_APPEND -append to an existing file. If the file does not exist, create the file.</li> </ul> </li> <li>• 03 - The file cannot be found.</li> <li>• 04 - The SQL_FILE_CREATE option was specified for a file with the same name as an existing file.</li> <li>• 05 - Access to the file was denied. The user does not have permission to open the file.</li> <li>• 06 - Access to the file was denied. The file is in use with incompatible modes. Files to be written to are opened in exclusive mode.</li> <li>• 07 - Disk full was encountered while writing to the file.</li> <li>• 08 - Unexpected end of file encountered while reading from the file.</li> <li>• 09 - A media error was encountered while accessing the file.</li> </ul>
54028	The maximum number of concurrent LOB handles has been reached.	<p>Maximum LOB locator assigned.</p> <p>The maximum number of concurrent LOB locators has been reached. A new locator can not be assigned.</p>
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<p><code>SQLFetch()</code> was called for an <i>StatementHandle</i> after <code>SQLExtendedFetch()</code> was called and before <code>SQLFreeStmt()</code> had been called with the <code>SQL_CLOSE</code> option.</p> <p>The function was called prior to calling <code>SQLPrepare()</code> or <code>SQLExecDirect()</code> for the <i>StatementHandle</i>.</p> <p>The function was called while in a data-at-execute (<code>SQLParamData()</code>, <code>SQLPutData()</code>) operation.</p> <p>The function was called while within a <code>BEGIN COMPOUND</code> and <code>END COMPOUND SQL</code> operation.</p>
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

## SQLFetch function (CLI) - Fetch next row

Table 61. SQLFetch SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Option type out of range.	The <i>FileOptions</i> argument of a previous SQLBindFileToCol() operation was not valid.
HYC00	Driver not capable.	CLI or the data source does not support the conversion specified by the combination of the <i>fCType</i> in SQLBindCol() or SQLBindFileToCol() and the SQL data type of the corresponding column.  A call to SQLBindCol() was made for a column data type which is not supported by CLI.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* fetch each row and display */
cliRC = SQLFetch(hstmt);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

if (cliRC == SQL_NO_DATA_FOUND)
{
    printf("\n Data not found.\n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
    printf("

/* fetch next row */
cliRC = SQLFetch(hstmt);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
}
```

## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLFetchScroll() fetches the specified rowset of data from the result set and returns data for all bound columns. Rowsets can be specified at an absolute or relative position or by bookmark.

### Syntax

```
SQLRETURN SQLFetchScroll (SQLHSTMT  
SQLSMALLINT SQLLEN  
StatementHandle,  
FetchOrientation,  
FetchOffset);
```



## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

### Function arguments

Table 62. SQLFetchScroll arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLUSMALLINT	<i>FetchOrientation</i>	input	Type of fetch: <ul style="list-style-type: none"><li>• SQL_FETCH_NEXT</li><li>• SQL_FETCH_PRIOR</li><li>• SQL_FETCH_FIRST</li><li>• SQL_FETCH_LAST</li><li>• SQL_FETCH_ABSOLUTE</li><li>• SQL_FETCH_RELATIVE</li><li>• SQL_FETCH_BOOKMARK</li></ul> For more information, see Positioning the Cursor.
SQLLEN	<i>FetchOffset</i>	input	Number of the row to fetch. The interpretation of this argument depends on the value of the <i>FetchOrientation</i> argument. For more information, see Positioning the Cursor.

### Usage

#### Overview

SQLFetchScroll() returns a specified rowset from the result set. Rowsets can be specified by absolute or relative position or by bookmark. SQLFetchScroll() can be called only while a result set exists, that is, after a call that creates a result set and before the cursor over that result set is closed. If any columns are bound, it returns the data in those columns. If the application has specified a pointer to a row status array or a buffer in which to return the number of rows fetched, SQLFetchScroll() returns this information as well. Calls to SQLFetchScroll() can be mixed with calls to SQLFetch() but cannot be mixed with calls to SQLExtendedFetch().

#### Positioning the cursor

When the result set is created, the cursor is positioned before the start of the result set. SQLFetchScroll() positions the block cursor based on the values of the *FetchOrientation* and *FetchOffset* arguments as shown in the following table. The exact rules for determining the start of the new rowset are shown in the next section.

#### FetchOrientation

##### Meaning

#### SQL\_FETCH\_NEXT

Return the next rowset. This is equivalent to calling SQLFetch(). SQLFetchScroll() ignores the value of *FetchOffset*.

#### SQL\_FETCH\_PRIOR

Return the prior rowset. SQLFetchScroll() ignores the value of *FetchOffset*.

#### SQL\_FETCH\_RELATIVE

Return the rowset *FetchOffset* from the start of the current rowset.

#### SQL\_FETCH\_ABSOLUTE

Return the rowset starting at row *FetchOffset*.

## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

### SQL\_FETCH\_FIRST

Return the first rowset in the result set. `SQLFetchScroll()` ignores the value of `FetchOffset`.

### SQL\_FETCH\_LAST

Return the last complete rowset in the result set. `SQLFetchScroll()` ignores the value of `FetchOffset`.

### SQL\_FETCH\_BOOKMARK

Return the rowset `FetchOffset` rows from the bookmark specified by the `SQL_ATTR_FETCH_BOOKMARK_PTR` statement attribute.

Not all cursors support all of these options. A static forward-only cursor, for example, will only support `SQL_FETCH_NEXT`. Scrollable cursors, such as keyset cursors, will support all of these options. The `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute specifies the number of rows in the rowset. If the rowset being fetched by `SQLFetchScroll()` overlaps the end of the result set, `SQLFetchScroll()` returns a partial rowset. That is, if  $S + R - 1$  is greater than  $L$ , where  $S$  is the starting row of the rowset being fetched,  $R$  is the rowset size, and  $L$  is the last row in the result set, then only the first  $L - S + 1$  rows of the rowset are valid. The remaining rows are empty and have a status of `SQL_ROW_NOROW`.

After `SQLFetchScroll()` returns, the rowset cursor is positioned on the first row of the result set.

### Returning data in bound columns

`SQLFetchScroll()` returns data in bound columns in the same way as `SQLFetch()`.

If no columns are bound, `SQLFetchScroll()` does not return data but does move the block cursor to the specified position. As with `SQLFetch()`, you can use `SQLGetData()` to retrieve the information in this case.

### Row status array

The row status array is used to return the status of each row in the rowset. The address of this array is specified with the `SQL_ATTR_ROW_STATUS_PTR` statement attribute. The array is allocated by the application and must have as many elements as are specified by the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute. Its values are set by `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()` (except when they have been called after the cursor has been positioned by `SQLExtendedFetch()`). If the value of the `SQL_ATTR_ROW_STATUS_PTR` statement attribute is a null pointer, these functions do not return the row status.

The contents of the row status array buffer are undefined if `SQLFetch()` or `SQLFetchScroll()` does not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`.

The following values are returned in the row status array.

#### Row status array value

##### Description

#### SQL\_ROW\_SUCCESS

The row was successfully fetched.

#### SQL\_ROW\_SUCCESS\_WITH\_INFO

The row was successfully fetched. However, a warning was returned about the row.

## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

### SQL\_ROW\_ERROR

An error occurred while fetching the row.

### SQL\_ROW\_ADDED

The row was inserted by SQLBulkOperations(). If the row is fetched again, or is refreshed by SQLSetPos() its status is SQL\_ROW\_SUCCESS.

This value is not set by SQLFetch() or SQLFetchScroll().

### SQL\_ROW\_UPDATED

The row was successfully fetched and has changed since it was last fetched from this result set. If the row is fetched again from this result set, or is refreshed by SQLSetPos(), the status changes to the row's new status.

### SQL\_ROW\_DELETED

The row has been deleted since it was last fetched from this result set.

### SQL\_ROW\_NOROW

The rowset overlapped the end of the result set and no row was returned that corresponded to this element of the row status array.

### Rows fetched buffer

The rows fetched buffer is used to return the number of rows fetched, including those rows for which no data was returned because an error occurred while they were being fetched. In other words, it is the number of rows for which the value in the row status array is not SQL\_ROW\_NOROW. The address of this buffer is specified with the SQL\_ATTR\_ROWS\_FETCHED\_PTR statement attribute. The buffer is allocated by the application. It is set by SQLFetch() and SQLFetchScroll(). If the value of the SQL\_ATTR\_ROWS\_FETCHED\_PTR statement attribute is a null pointer, these functions do not return the number of rows fetched. To determine the number of the current row in the result set, an application can call SQLGetStmtAttr() with the SQL\_ATTR\_ROW\_NUMBER attribute.

The contents of the rows fetched buffer are undefined if SQLFetch() or SQLFetchScroll() does not return SQL\_SUCCESS or SQL\_SUCCESS\_WITH\_INFO, except when SQL\_NO\_DATA is returned, in which case the value in the rows fetched buffer is set to 0.

### Error handling

SQLFetchScroll() returns errors and warnings in the same manner as SQLFetch().

### Descriptors and SQLFetchScroll()

SQLFetchScroll() interacts with descriptors in the same manner as SQLFetch().

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_NO\_DATA
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

### Diagnostics

The return code associated with each SQLSTATE value is SQL\_ERROR, unless noted otherwise. If an error occurs on a single column, SQLGetDiagField() can be called with a *DiagIdentifier* of SQL\_DIAG\_COLUMN\_NUMBER to determine the column the error occurred on; and SQLGetDiagField() can be called with a *DiagIdentifier* of SQL\_DIAG\_ROW\_NUMBER to determine the row containing that column.

Table 63. SQLFetchScroll SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	String or binary data returned for a column resulted in the truncation of non-blank character or non-NULL binary data. String values are right truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
01S01	Error in row.	An error occurred while fetching one or more rows. (Function returns SQL_SUCCESS_WITH_INFO.) (This SQLSTATE is only returned when connected to CLI v2.)
01S06	Attempt to fetch before the result set returned the first rowset.	The requested rowset overlapped the start of the result set when the current position was beyond the first row, and either <i>FetchOrientation</i> was SQL_PRIOR, or <i>FetchOrientation</i> was SQL_RELATIVE with a negative <i>FetchOffset</i> whose absolute value was less than or equal to the current SQL_ATTR_ROW_ARRAY_SIZE. (Function returns SQL_SUCCESS_WITH_INFO.)
01S07	Fractional truncation.	The data returned for a column was truncated. For numeric data types, the fractional part of the number was truncated. For time or timestamp data types, the fractional portion of the time was truncated.
07002	Too many columns.	A column number specified in the binding for one or more columns was greater than the number of columns in the result set.
07006	Invalid conversion.	A data value of a column in the result set could not be converted to the C data type specified by <i>TargetType</i> in SQLBindCol().
07009	Invalid descriptor index.	Column 0 was bound and the SQL_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
22001	String data right truncation.	A variable-length bookmark returned for a row was truncated.
22002	Invalid output or indicator buffer specified.	NULL data was fetched into a column whose <i>StrLen_or_IndPtr</i> set by SQLBindCol() (or SQL_DESC_INDICATOR_PTR set by SQLSetDescField() or SQLSetDescRec()) was a null pointer.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for one or more bound columns would have caused the whole (as opposed to fractional) part of the number to be truncated.
22007	Invalid datetime format.	A character column in the result set was bound to a date, time, or timestamp C structure, and a value in the column was, respectively, an invalid date, time, or timestamp.
22012	Division by zero is invalid.	A value from an arithmetic expression was returned which resulted in division by zero.

## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

Table 63. SQLFetchScroll SQLSTATEs (continued)

SQLSTATE	Description	Explanation
22018	Invalid character value for cast specification.	A character column in the result set was bound to a character C buffer and the column contained a character for which there was no representation in the character set of the buffer. A character column in the result set was bound to an approximate numeric C buffer and a value in the column could not be cast to a valid approximate numeric value. A character column in the result set was bound to an exact numeric C buffer and a value in the column could not be cast to a valid exact numeric value. A character column in the result set was bound to a datetime C buffer and a value in the column could not be cast to a valid datetime value.
24000	Invalid cursor state.	The <i>StatementHandle</i> was in an executed state but no result set was associated with the <i>StatementHandle</i> .
40001	Transaction rollback.	The transaction in which the fetch was executed was terminated to prevent deadlock.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect(), SQLExecute(), or a catalog function.  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.  SQLExecute() or SQLExecDirect() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.  SQLFetchScroll() was called for a <i>StatementHandle</i> after SQLExtendedFetch() was called and before SQLFreeStmt() with SQL_CLOSE was called.
HY106	Fetch type out of range.	The value specified for the argument <i>FetchOrientation</i> was invalid.  The argument <i>FetchOrientation</i> was SQL_FETCH_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.  The value of the SQL_CURSOR_TYPE statement attribute was SQL_CURSOR_FORWARD_ONLY and the value of argument <i>FetchOrientation</i> was not SQL_FETCH_NEXT.

## SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

Table 63. SQLFetchScroll SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY107	Row value out of range.	The value specified with the SQL_ATTR_CURSOR_TYPE statement attribute was SQL_CURSOR_KEYSET_DRIVEN, but the value specified with the SQL_ATTR_KEYSET_SIZE statement attribute was greater than 0 and less than the value specified with the SQL_ATTR_ROW_ARRAY_SIZE statement attribute.
HY111	Invalid bookmark value.	The argument <i>FetchOrientation</i> was SQL_FETCH_BOOKMARK and the bookmark pointed to by the value in the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute was not valid or was a null pointer.
HYC00	Driver not capable.	The specified fetch type is not supported.  The conversion specified by the combination of the <i>TargetType</i> in SQLBindCol() and the SQL data type of the corresponding column is not supported.

### Restrictions

None.

### Example

```
/* fetch the rowset: row15, row16, row17, row18, row19 */
printf("\n Fetch the rowset: row15, row16, row17, row18, row19.\n");

/* fetch the rowset and return data for all bound columns */
cliRC = SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 15);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* call SQLFetchScroll with SQL_FETCH_RELATIVE offset 3 */
printf(" SQLFetchScroll with SQL_FETCH_RELATIVE offset 3.\n");
printf("  COL1          COL2          \n");
printf("  -----  ----- \n");

/* fetch the rowset and return data for all bound columns */
cliRC = SQLFetchScroll(hstmt, SQL_FETCH_RELATIVE, 3);
```

## Cursor positioning rules for SQLFetchScroll() (CLI)

The following sections describe the exact rules for each value of *FetchOrientation*. These rules use the following notation:

### FetchOrientation

#### Meaning

#### Before start

The block cursor is positioned before the start of the result set. If the first row of the new rowset is before the start of the result set, SQLFetchScroll() returns SQL\_NO\_DATA.

#### After end

The block cursor is positioned after the end of the result set. If the first row of the new rowset is after the end of the result set, SQLFetchScroll() returns SQL\_NO\_DATA.

#### CurrRowsetStart

The number of the first row in the current rowset.

## Cursor positioning rules for SQLFetchScroll() (CLI)

### LastResultRow

The number of the last row in the result set.

### RowsetSize

The rowset size.

### FetchOffset

The value of the *FetchOffset* argument.

### BookmarkRow

The row corresponding to the bookmark specified by the SQL\_ATTR\_FETCH\_BOOKMARK\_PTR statement attribute.

### SQL\_FETCH\_NEXT rules:

Table 64. SQL\_FETCH\_NEXT rules:

Condition	First row of new rowset
Before start	1
$\text{CurrRowsetStart} + \text{RowsetSize} \leq \text{LastResultRow}$	$\text{CurrRowsetStart} + \text{RowsetSize}$
$\text{CurrRowsetStart} + \text{RowsetSize} > \text{LastResultRow}$	After end
After end	After end

### SQL\_FETCH\_PRIOR rules:

Table 65. SQL\_FETCH\_PRIOR rules:

Condition	First row of new rowset
Before start	Before start
$\text{CurrRowsetStart} = 1$	Before start
$1 < \text{CurrRowsetStart} \leq \text{RowsetSize}$	1 <sup>a</sup>
$\text{CurrRowsetStart} > \text{RowsetSize}$	$\text{CurrRowsetStart} - \text{RowsetSize}$
After end AND $\text{LastResultRow} < \text{RowsetSize}$	1 <sup>a</sup>
After end AND $\text{LastResultRow} \geq \text{RowsetSize}$	$\text{LastResultRow} - \text{RowsetSize} + 1$

- a SQLFetchScroll() returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset.) and SQL\_SUCCESS\_WITH\_INFO.

### SQL\_FETCH\_RELATIVE rules:

Table 66. SQL\_FETCH\_RELATIVE rules:

Condition	First row of new rowset
(Before start AND $\text{FetchOffset} > 0$ ) OR (After end AND $\text{FetchOffset} = 0$ )	-- <sup>a</sup>
Before start AND $\text{FetchOffset} \leq 0$	Before start
$\text{CurrRowsetStart} = 1$ AND $\text{FetchOffset} < 0$	Before start
$\text{CurrRowsetStart} > 1$ AND $\text{CurrRowsetStart} + \text{FetchOffset} < 1$ AND $ \text{FetchOffset}  > \text{RowsetSize}$	Before start
$\text{CurrRowsetStart} > 1$ AND $\text{CurrRowsetStart} + \text{FetchOffset} < 1$ AND $ \text{FetchOffset}  \leq \text{RowsetSize}$	1 <sup>b</sup>
$1 \leq \text{CurrRowsetStart} + \text{FetchOffset} \leq \text{LastResultRow}$	$\text{CurrRowsetStart} + \text{FetchOffset}$
$\text{CurrRowsetStart} + \text{FetchOffset} > \text{LastResultRow}$	After end

## Cursor positioning rules for SQLFetchScroll() (CLI)

Table 66. SQL\_FETCH\_RELATIVE rules: (continued)

Condition	First row of new rowset
After end AND <i>FetchOffset</i> >= 0	After end

- **a** SQLFetchScroll() returns the same rowset as if it was called with *FetchOrientation* set to SQL\_FETCH\_ABSOLUTE. For more information, see the SQL\_FETCH\_ABSOLUTE section.
- **b** SQLFetchScroll() returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset.) and SQL\_SUCCESS\_WITH\_INFO.

### SQL\_FETCH\_ABSOLUTE rules:

Table 67. SQL\_FETCH\_ABSOLUTE rules:

Condition	First row of new rowset
<i>FetchOffset</i> < 0 AND   <i>FetchOffset</i>   <= LastResultRow	LastResultRow + <i>FetchOffset</i> + 1
<i>FetchOffset</i> < 0 AND   <i>FetchOffset</i>   > LastResultRow AND   <i>FetchOffset</i>   > RowsetSize	Before start
<i>FetchOffset</i> < 0 AND   <i>FetchOffset</i>   > LastResultRow AND   <i>FetchOffset</i>   <= RowsetSize	1 <sup>a</sup>
<i>FetchOffset</i> = 0	Before start
1 <= <i>FetchOffset</i> <= LastResultRow	<i>FetchOffset</i>
<i>FetchOffset</i> > LastResultRow	After end

- **a** SQLFetchScroll() returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset.) and SQL\_SUCCESS\_WITH\_INFO.

### SQL\_FETCH\_FIRST rules:

Table 68. SQL\_FETCH\_FIRST rules:

Condition	First row of new rowset
Any	1

### SQL\_FETCH\_LAST rules:

Table 69. SQL\_FETCH\_LAST rules:

Condition	First row of new rowset
RowsetSize = LastResultRow	LastResultRow - RowsetSize + 1
RowsetSize > LastResultRow	1

### SQL\_FETCH\_BOOKMARK rules:

Table 70. SQL\_FETCH\_BOOKMARK rules:

Condition	First row of new rowset
BookmarkRow + <i>FetchOffset</i> < 1	Before start
1 <= BookmarkRow + <i>FetchOffset</i> <= LastResultRow	BookmarkRow + <i>FetchOffset</i>
BookmarkRow + <i>FetchOffset</i> > LastResultRow	After end



## SQLForeignKeys function (CLI) - Get the list of foreign key columns

Returns information about foreign keys for the specified table.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

The SQLForeignKeys() function returns information about foreign keys for the specified table. The information is returned in an SQL result set which you can process by using the same functions that you use to retrieve a result that is generated by a query.

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is SQLForeignKeysW(). See “Unicode functions (CLI)” on page 5 for information about ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLForeignKeys (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLCHAR           *PKCatalogName, /* szPkCatalogName */
    SQLSMALLINT      NameLength1,    /* cbPkCatalogName */
    SQLCHAR           *PKSchemaName, /* szPkSchemaName */
    SQLSMALLINT      NameLength2,    /* cbPkSchemaName */
    SQLCHAR           *PKTableName,  /* szPkTableName */
    SQLSMALLINT      NameLength3,    /* cbPkTableName */
    SQLCHAR           *FKCatalogName, /* szFkCatalogName */
    SQLSMALLINT      NameLength4,    /* cbFkCatalogName */
    SQLCHAR           *FKSchemaName, /* szFkSchemaName */
    SQLSMALLINT      NameLength5,    /* cbFkSchemaName */
    SQLCHAR           *FKTableName,  /* szFkTableName */
    SQLSMALLINT      NameLength6); /* cbFkTableName */
```

### Function arguments

Table 71. SQLForeignKeys arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>PKCatalogName</i>	Input	The catalog qualifier of the 3-part primary key table name. If the target DBMS does not support 3-part naming, and <i>PKCatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS is returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>PKCatalogName</i> , or SQL_NTS if <i>PKCatalogName</i> is null-terminated.
SQLCHAR *	<i>PKSchemaName</i>	Input	The schema qualifier of the primary key table.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>PKSchemaName</i> , or SQL_NTS if <i>PKSchemaName</i> is null-terminated.
SQLCHAR *	<i>PKTableName</i>	Input	The name of the table name that contains the primary key.

## SQLForeignKeys function (CLI) - Get the list of foreign key columns

Table 71. SQLForeignKeys arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>PKTableName</i> , or SQL_NTS if <i>PKTableName</i> is null-terminated.
SQLCHAR *	<i>FKCatalogName</i>	Input	The catalog qualifier of the 3-part foreign key table name. If the target DBMS does not support 3-part naming, and <i>FKCatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS is returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	<i>NameLength4</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>FKCatalogName</i> , or SQL_NTS if <i>FKCatalogName</i> is null-terminated.
SQLCHAR *	<i>FKSchemaName</i>	Input	The schema qualifier of the table that contains the foreign key.
SQLSMALLINT	<i>NameLength5</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>FKSchemaName</i> , or SQL_NTS if <i>FKSchemaName</i> is null-terminated.
SQLCHAR *	<i>FKTableName</i>	Input	The name of the table that contains the foreign key.
SQLSMALLINT	<i>NameLength6</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>FKTableName</i> , or SQL_NTS if <i>FKTableName</i> is null-terminated.

### Usage

If *PKTableName* contains a table name, and *FKTableName* is an empty string, the SQLForeignKeys() function returns a result set that contains the primary key of the specified table and all of the foreign keys (in other tables) that refer to it.

If *FKTableName* contains a table name, and *PKTableName* is an empty string, the SQLForeignKeys() function returns a result set that contains all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both *PKTableName* and *FKTableName* contain table names, the SQLForeignKeys() function returns the foreign keys in the table that are specified in *FKTableName*, which refer to the primary key of the table that is specified in *PKTableName*. There should be one key at the most.

If the schema qualifier argument that is associated with a table name is not specified, the schema name defaults to the table name that is currently in effect for the current connection.

Columns Returned by SQLForeignKeys lists the columns of the result set that is generated by the SQLForeignKeys() call. If the foreign keys that are associated with a primary key are requested, the result set is ordered by FKTABLE\_CAT, FKTABLE\_SCHEM, FKTABLE\_NAME, and ORDINAL\_POSITION. If the primary

## SQLForeignKeys function (CLI) - Get the list of foreign key columns

keys that are associated with a foreign key are requested, the result set is ordered by PKTABLE\_CAT, PKTABLE\_SCHEM, PKTABLE\_NAME, and ORDINAL\_POSITION.

Call SQLGetInfo() with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to respectively determine the actual lengths of the associated TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

In Version 9.7 Fix Pack 5, you can specify \*ALL as a value in the *SchemaName* to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns that are returned by SQLForeignKeys

#### Column 1 PKTABLE\_CAT (VARCHAR(128))

Name of the catalog for PKTABLE\_NAME. The value is NULL if this table does not have catalogs.

#### Column 2 PKTABLE\_SCHEM (VARCHAR(128))

Name of the schema containing PKTABLE\_NAME.

#### Column 3 PKTABLE\_NAME (VARCHAR(128) not NULL)

Name of the table containing the primary key.

#### Column 4 PKCOLUMN\_NAME (VARCHAR(128) not NULL)

Primary key column name.

#### Column 5 FKTABLE\_CAT (VARCHAR(128))

Name of the catalog for FKTABLE\_NAME. The value is NULL if this table does not have catalogs.

#### Column 6 FKTABLE\_SCHEM (VARCHAR(128))

Name of the schema containing FKTABLE\_NAME.

#### Column 7 FKTABLE\_NAME (VARCHAR(128) not NULL)

Name of the table containing the foreign key.

#### Column 8 FKCOLUMN\_NAME (VARCHAR(128) not NULL)

Foreign key column name.

#### Column 9 KEY\_SEQ (SMALLINT not NULL)

Ordinal position of the column in the key, starting at 1.

#### Column 10 UPDATE\_RULE (SMALLINT)

Action to be applied to the foreign key when the SQL operation is UPDATE:

- SQL\_RESTRICT
- SQL\_NO\_ACTION

The update rule for IBM DB2 DBMSs is always either RESTRICT or SQL\_NO\_ACTION. However, ODBC applications might encounter the listed UPDATE\_RULE values when connected to non-IBM RDBMSs:

- SQL\_CASCADE

## SQLForeignKeys function (CLI) - Get the list of foreign key columns

- SQL\_SET\_NULL

### Column 11 DELETE\_RULE (SMALLINT)

Action to be applied to the foreign key when the SQL operation is DELETE:

- SQL\_CASCADE
- SQL\_NO\_ACTION
- SQL\_RESTRICT
- SQL\_SET\_DEFAULT
- SQL\_SET\_NULL

### Column 12 FK\_NAME (VARCHAR(128))

Foreign key identifier. NULL if not applicable to the data source.

### Column 13 PK\_NAME (VARCHAR(128))

Primary key identifier. NULL if not applicable to the data source.

### Column 14 DEFERRABILITY (SMALLINT)

One of:

- SQL\_INITIALLY\_DEFERRED
- SQL\_INITIALLY\_IMMEDIATE
- SQL\_NOT\_DEFERRABLE

**Note:** The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the SQLForeignKeys() result set in ODBC.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 72. SQLForeignKeys SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor is already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	The arguments <i>PKTableName</i> and <i>FKTableName</i> are both NULL pointers.

## SQLForeignKeys function (CLI) - Get the list of foreign key columns

Table 72. SQLForeignKeys SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while in a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	<p>The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.</p> <p>The length of the table or owner name is greater than the maximum length that is supported by the server.</p>
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* get the list of foreign key columns */
cliRC = SQLForeignKeys(hstmt,
                      NULL,
                      0,
                      tbSchema,
                      SQL_NTS,
                      tbName,
                      SQL_NTS,
                      NULL,
                      0,
                      NULL,
                      SQL_NTS,
                      NULL,
                      SQL_NTS);
```

## SQLFreeConnect function (CLI) - Free connection handle

### Deprecated

#### Note:

In ODBC 3.0, SQLFreeConnect() has been deprecated and replaced with SQLFreeHandle().

Although this version of CLI continues to support SQLFreeConnect(), use SQLFreeHandle() in your CLI programs so that they conform to the latest standards.

## SQLFreeConnect function (CLI) - Free connection handle

### Migrating to the new function

The statement:

```
SQLFreeConnect(hdbc);
```

for example, would be rewritten using the new function as:

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

---

## SQLFreeEnv function (CLI) - Free environment handle

### Deprecated

**Note:**

In ODBC 3.0, `SQLFreeEnv()` has been deprecated and replaced with `SQLFreeHandle()`.

Although this version of CLI continues to support `SQLFreeEnv()`, use `SQLFreeHandle()` in your CLI programs so that they conform to the latest standards.

### Migrating to the new function

The statement:

```
SQLFreeEnv(henv);
```

for example, would be rewritten using the new function as:

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

---

## SQLFreeHandle function (CLI) - Free handle resources

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

`SQLFreeHandle()` frees resources associated with a specific environment, connection, statement, or descriptor handle.

**Note:** This function is a generic function for freeing resources. It replaces the ODBC 2.0 functions `SQLFreeConnect()` (for freeing a connection handle), and `SQLFreeEnv()` (for freeing an environment handle). `SQLFreeHandle()` also replaces the ODBC 2.0 function `SQLFreeStmt()` (with the `SQL_DROP` Option) for freeing a statement handle.

### Syntax

```
SQLRETURN SQLFreeHandle (
    SQLSMALLINT HandleType, /* fHandleType */
    SQLHANDLE Handle);      /* hHandle */
```

## Function arguments

Table 73. SQLFreeHandle arguments

Data type	Argument	Use	Description
SQLSMALLINT	<i>HandleType</i>	input	The type of handle to be freed by SQLFreeHandle(). Must be one of the following values: <ul style="list-style-type: none"> <li>SQL_HANDLE_ENV</li> <li>SQL_HANDLE_DBC</li> <li>SQL_HANDLE_STMT</li> <li>SQL_HANDLE_DESC</li> </ul> If <i>HandleType</i> is not one of the above values, SQLFreeHandle() returns SQL_INVALID_HANDLE.
SQLHANDLE	<i>Handle</i>	input	The handle to be freed.

## Usage

SQLFreeHandle() is used to free handles for environments, connections, statements, and descriptors.

An application should not use a handle after it has been freed; CLI does not check the validity of a handle in a function call.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

If SQLFreeHandle() returns SQL\_ERROR, the handle is still valid.

## Diagnostics

Table 74. SQLFreeHandle SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The <i>HandleType</i> argument was SQL_HANDLE_DBC, and the communication link between CLI and the data source to which it was trying to connect failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

## SQLFreeHandle function (CLI) - Free handle resources

Table 74. SQLFreeHandle SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The <i>HandleType</i> argument was SQL_HANDLE_ENV, and at least one connection was in an allocated or connected state. SQLDisconnect() and SQLFreeHandle() with a <i>HandleType</i> of SQL_HANDLE_DBC must be called for each connection before calling SQLFreeHandle() with a <i>HandleType</i> of SQL_HANDLE_ENV. The <i>HandleType</i> argument was SQL_HANDLE_DBC, and the function was called before calling SQLDisconnect() for the connection.</p> <p>The <i>HandleType</i> argument was SQL_HANDLE_STMT; an asynchronously executing function was called on the statement handle; and the function was still executing when this function was called.</p> <p>The <i>HandleType</i> argument was SQL_HANDLE_STMT; SQLExecute() or SQLExecDirect() was called with the statement handle, and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. (DM) All subsidiary handles and other resources were not released before SQLFreeHandle() was called.</p>
HY013	Unexpected memory handling error.	The <i>HandleType</i> argument was SQL_HANDLE_STMT or SQL_HANDLE_DESC, and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY017	Invalid use of an automatically allocated descriptor handle.	The <i>Handle</i> argument was set to the handle for an automatically allocated descriptor or an implementation descriptor.

### Restrictions

None.

### Example

```
/* free the statement handle */
cliRC = SQLFreeHandle(SQL_HANDLE_STMT, hstmt2);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_STMT,
                                        hstmt2,
                                        cliRC,
                                        henv,
                                        hdbc,
                                        pOutSqlrc,
                                        outMsg,
                                        "SQLFreeHandle");

/* ... */
/* free the database handle */
cliRC = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_DBC,
                                        hdbc,
                                        cliRC,
                                        henv,
                                        hdbc,
                                        pOutSqlrc,
                                        outMsg,
                                        "SQLFreeHandle");

/* free the environment handle */
cliRC = SQLFreeHandle(SQL_HANDLE_ENV, henv);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_ENV,
                                        henv,
```



```
cliRC,
henv,
hdbc,
pOutSqlrc,
outMsg,
"SQLFreeHandle");
```

## SQLFreeStmt function (CLI) - Free (or reset) a statement handle

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLFreeStmt() ends processing on the statement referenced by the statement handle. Use this function to:

- Close a cursor and discard all pending results
- Disassociate (reset) parameters from application variables and LOB file references
- Unbind columns from application variables and LOB file references
- Drop the statement handle and free the CLI resources associated with the statement handle.

SQLFreeStmt() is called after executing an SQL statement and processing the results.

### Syntax

```
SQLRETURN SQLFreeStmt (SQLHSTMT StatementHandle, /* hstmt */
                        SQLUSMALLINT Option); /* fOption */
```

### Function arguments

Table 75. SQLFreeStmt arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLUSMALLINT	<i>Option</i>	input	Option which specifies the manner of freeing the statement handle. The option must have one of the following values: <ul style="list-style-type: none"> <li>• SQL_CLOSE</li> <li>• SQL_DROP</li> <li>• SQL_UNBIND</li> <li>• SQL_RESET_PARAMS</li> </ul>

### Usage

SQLFreeStmt() can be called with the following options:

#### SQL\_CLOSE

The cursor (if any) associated with the statement handle (*StatementHandle*) is closed and all pending results are discarded. The application can reopen the cursor by calling SQLExecute() with the same or different values in the application variables (if any) that are bound to *StatementHandle*. The cursor name is retained until the statement handle is dropped or a subsequent call to SQLGetCursorName() is successful. If no cursor has been associated with the statement handle, this option has no effect (no warning or error is generated).

## SQLFreeStmt function (CLI) - Free (or reset) a statement handle

SQLCloseCursor() can also be used to close a cursor.

### SQL\_DROP

CLI resources associated with the input statement handle are freed, and the handle is invalidated. The open cursor, if any, is closed and all pending results are discarded.

This option has been replaced with a call to SQLFreeHandle() with the *HandleType* set to SQL\_HANDLE\_STMT. Although this version of CLI continues to support this option, begin using SQLFreeHandle() in your CLI programs so that they conform to the latest standards.

### SQL\_UNBIND

Sets the SQL\_DESC\_COUNT field of the ARD (Application Row Descriptor) to 0, releasing all column buffers bound by SQLBindCol() or SQLBindFileToCol() for the given *StatementHandle*. This does not unbind the bookmark column; to do that, the SQL\_DESC\_DATA\_PTR field of the ARD for the bookmark column is set to NULL. Note that if this operation is performed on an explicitly allocated descriptor that is shared by more than one statement, the operation will affect the bindings of all statements that share the descriptor.

### SQL\_RESET\_PARAMS

Sets the SQL\_DESC\_COUNT field of the APD (Application Parameter Descriptor) to 0, releasing all parameter buffers set by SQLBindParameter() or SQLBindFileToParam() for the given *StatementHandle*. Note that if this operation is performed on an explicitly allocated descriptor that is shared by more than one statement, this operation will affect the bindings of all the statements that share the descriptor.

SQLFreeStmt() has no effect on LOB locators, call SQLExecDirect() with the FREE LOCATOR statement to free a locator.

It is possible to reuse a statement handle to execute a different statement:

- If the handle was associated with a query, catalog function or SQLGetTypeInfo(), you must close the cursor.
- If the handle was bound with a different number or type of parameters, the parameters must be reset.
- If the handle was bound with a different number or type of column bindings, the columns must be unbound.

Alternatively you may drop the statement handle and allocate a new one.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

SQL\_SUCCESS\_WITH\_INFO is not returned if *Option* is set to SQL\_DROP, as there would be no statement handle to use when SQLGetDiagRec() or SQLGetDiagField() is called.

## SQLFreeStmt function (CLI) - Free (or reset) a statement handle

### Diagnostics

Table 76. SQLFreeStmt SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.
HY092	Option type out of range.	The value specified for the argument <i>Option</i> was not SQL_CLOSE, SQL_DROP, SQL_UNBIND, or SQL_RESET_PARAMS.
HY506	Error closing a file.	Error encountered while trying to close a temporary file.

### Authorization

None.

### Example

```
/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_UNBIND);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
    return 1;
}

/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_RESET_PARAMS);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
    return 1;
}

/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_CLOSE);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
    return 1;
}
```

## SQLGetConnectAttr function (CLI) - Get current attribute setting

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLGetConnectAttr() returns the current setting of a connection attribute.

## SQLGetConnectAttr function (CLI) - Get current attribute setting

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetConnectAttrW()`. Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLGetConnectAttr(SQLHDBC          ConnectionHandle,
                             SQLINTEGER      Attribute,
                             SQLPOINTER     ValuePtr,
                             SQLINTEGER     BufferLength,
                             SQLINTEGER     *StringLengthPtr);
```

### Function arguments

Table 77. *SQLGetConnectAttr* arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle.
SQLINTEGER	<i>Attribute</i>	input	<i>Attribute</i> to retrieve.
SQLPOINTER	<i>ValuePtr</i>	output	A pointer to memory in which to return the current value of the attribute specified by <i>Attribute</i> .
SQLINTEGER	<i>BufferLength</i>	input	<ul style="list-style-type: none"> <li>If <i>ValuePtr</i> points to a character string, this argument should be the length of <i>*ValuePtr</i>.</li> <li>If <i>ValuePtr</i> is a pointer, but not to a string, then <i>BufferLength</i> should have the value <code>SQL_IS_POINTER</code>.</li> <li>If the value in <i>*ValuePtr</i> is a Unicode string the <i>BufferLength</i> argument must be an even number.</li> </ul>
SQLINTEGER *	<i>StringLengthPtr</i>	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in <i>*ValuePtr</i> . If <i>ValuePtr</i> is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than <i>BufferLength</i> minus the length of the null-termination character, the data in <i>*ValuePtr</i> is truncated to <i>BufferLength</i> minus the length of the null-termination character and is null-terminated by CLI.

### Usage

If *Attribute* specifies an attribute that returns a string, *ValuePtr* must be a pointer to a buffer for the string. The maximum length of the string, including the null termination character, will be *BufferLength* bytes.

Depending on the attribute, an application does not need to establish a connection prior to calling `SQLGetConnectAttr()`. However, if `SQLGetConnectAttr()` is called and the specified attribute does not have a default and has not been set by a prior call to `SQLSetConnectAttr()`, `SQLGetConnectAttr()` will return `SQL_NO_DATA`.

If *Attribute* is `SQL_ATTR_TRACE` or `SQL_ATTR_TRACEFILE`, *ConnectionHandle* does not have to be valid, and `SQLGetConnectAttr()` will not return `SQL_ERROR` if *ConnectionHandle* is invalid. These attributes apply to all connections. `SQLGetConnectAttr()` will return `SQL_ERROR` if another argument is invalid.

## SQLGetConnectAttr function (CLI) - Get current attribute setting

While an application can set statement attributes using `SQLSetConnectAttr()`, an application cannot use `SQLGetConnectAttr()` to retrieve statement attribute values; it must call `SQLGetStmtAttr()` to retrieve the setting of statement attributes.

The `SQL_ATTR_AUTO_IPD` connection attribute can be returned by a call to `SQLGetConnectAttr()`, but cannot be set by a call to `SQLSetConnectAttr()`.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_NO_DATA`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 78. `SQLGetConnectAttr` SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated.	The data returned in <code>*ValuePtr</code> was truncated to be <code>BufferLength</code> minus the length of a null termination character. The length of the untruncated string value is returned in <code>*StringLengthPtr</code> . (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection is closed.	An <code>Attribute</code> value was specified that required an open connection.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by <code>SQLGetDiagRec()</code> in the <code>*MessageText</code> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	<code>SQLBrowseConnect()</code> was called for the <code>ConnectionHandle</code> and returned <code>SQL_NEED_DATA</code> . This function was called before <code>SQLBrowseConnect()</code> returned <code>SQL_SUCCESS_WITH_INFO</code> or <code>SQL_SUCCESS</code> .
HY090	Invalid string or buffer length.	The value specified for the argument <code>BufferLength</code> was less than 0.
HY092	Option type out of range.	The value specified for the argument <code>Attribute</code> was not valid.
HYC00	Driver not capable.	The value specified for the argument <code>Attribute</code> was a valid connection or statement attribute for the version of the CLI driver, but was not supported by the data source.

### Restrictions

None.

### Example

```
SQLINTEGER autocommit;

/* ... */

/* get the current setting for the AUTOCOMMIT attribute */
cliRC = SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, &autocommit, 0, NULL);
```

---

### SQLGetConnectOption function (CLI) - Return current setting of a connect option

#### Deprecated

#### Note:

In ODBC version 3, SQLGetConnectOption() has been deprecated and replaced with SQLGetConnectAttr().

Although this version of CLI continues to support SQLGetConnectOption(), use SQLGetConnectAttr() in your CLI programs so that they conform to the latest standards.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetConnectOptionW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

#### Migrating to the new function

The statement:

```
SQLGetConnectOption(hdbc, SQL_ATTR_AUTOCOMMIT, pvAutoCommit);
```

for example, would be rewritten using the new function as:

```
SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, pvAutoCommit,  
SQL_IS_POINTER, NULL);
```

---

### SQLGetCursorName function (CLI) - Get cursor name

#### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLGetCursorName() returns the cursor name associated with the input statement handle. If a cursor name was explicitly set by calling SQLSetCursorName(), this name will be returned; otherwise, an implicitly generated name will be returned.

**Unicode Equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetCursorNameW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

#### Syntax

```
SQLRETURN SQLGetCursorName (  
    SQLHSTMT StatementHandle, /* hstmt */  
    SQLCHAR *CursorName, /* szCursor */  
    SQLSMALLINT BufferLength, /* cbCursorMax */  
    SQLSMALLINT *NameLengthPtr); /* pcbCursor */
```

## Function arguments

Table 79. SQLGetCursorName arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLCHAR *	<i>CursorName</i>	output	Cursor name
SQLSMALLINT	<i>BufferLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>CursorName</i> .
SQLSMALLINT *	<i>NameLengthPtr</i>	output	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return for <i>CursorName</i> .

## Usage

SQLGetCursorName() will return the cursor name set explicitly with SQLSetCursorName(), or if no name was set, it will return the cursor name internally generated by CLI. If SQLGetCursorName() is called before a statement has been prepared on the input statement handle, an error will result. The internal cursor name is generated on a statement handle the first time dynamic SQL is prepared on the statement handle, not when the handle is allocated.

If a name is set explicitly using SQLSetCursorName(), this name will be returned until the statement is dropped, or until another explicit name is set.

Internally generated cursor names always begin with SQLCUR or SQL\_CUR. Cursor names are always 128 SQLCHAR or SQLWCHAR elements or less, and are always unique within a connection.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 80. SQLGetCursorName SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated.	The cursor name returned in <i>CursorName</i> was longer than the value in <i>BufferLength</i> , and is truncated to <i>BufferLength</i> - 1 bytes. The argument <i>NameLengthPtr</i> contains the length of the full cursor name available for return. The function returns SQL_SUCCESS_WITH_INFO.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

## SQLGetCursorName function (CLI) - Get cursor name

Table 80. SQLGetCursorName SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called For the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument <i>BufferLength</i> is less than 0.

### Restrictions

ODBC generated cursor names start with SQL\_CUR, CLI generated cursor names start with SQLCUR, and X/Open CLI generated cursor names begin with either SQLCUR or SQL\_CUR.

### Example

```
SQLCHAR cursorName[20];

/* ... */

/* get the cursor name of the SELECT statement */
cliRC = SQLGetCursorName(hstmtSelect, cursorName, 20, &cursorLen);
```

## SQLGetData function (CLI) - Get data from a column

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLGetData() retrieves data for a single column in the current row of the result set. This is an alternative to SQLBindCol(), which is used to transfer data directly into application variables or LOB locators on each SQLFetch() or SQLFetchScroll() call. An application can either bind LOBs with SQLBindCol() or use SQLGetData() to retrieve LOBs, but both methods cannot be used together. SQLGetData() can also be used to retrieve large data values in pieces.

SQLFetch() or SQLFetchScroll() must be called before SQLGetData().

After calling SQLGetData() for each column, SQLFetch() or SQLFetchScroll() is called to retrieve the next row.

### Syntax

```
SQLRETURN SQLGetData (
    SQLHSTMT      StatementHandle, /* hstmt */
    SQLUSMALLINT ColumnNumber,     /* icol */
    SQLSMALLINT   TargetType,       /* fctype */
```



## SQLGetData function (CLI) - Get data from a column

```

SQLPOINTER TargetValuePtr, /* rgbValue */
SQLLEN BufferLength, /* cbValueMax */
SQLLEN *StrLen_or_IndPtr); /* pcbValue */

```

### Function arguments

Table 81. SQLGetData arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLUSMALLINT	<i>ColumnNumber</i>	input	Column number for which the data retrieval is requested. Result set columns are numbered sequentially from left to right. <ul style="list-style-type: none"> <li>Column numbers start at 1 if bookmarks are not used (SQL_ATTR_USE_BOOKMARKS statement attribute set to SQL_UB_OFF).</li> <li>Column numbers start at 0 if bookmarks are used (the statement attribute set to SQL_UB_ON or SQL_UB_VARIABLE).</li> </ul>
SQLSMALLINT	<i>TargetType</i>	input	The C data type of the column identifier by <i>ColumnNumber</i> . The following types are supported: <ul style="list-style-type: none"> <li>SQL_C_BINARY</li> <li>SQL_C_BIT</li> <li>SQL_C_BLOB_LOCATOR</li> <li>SQL_C_CHAR</li> <li>SQL_C_CLOB_LOCATOR</li> <li>SQL_C_DBCHAR</li> <li>SQL_C_DBCLOB_LOCATOR</li> <li>SQL_C_DECIMAL_IBM</li> <li>SQL_C_DOUBLE</li> <li>SQL_C_FLOAT</li> <li>SQL_C_LONG</li> <li>SQL_C_NUMERIC<sup>a</sup></li> <li>SQL_C_SBIGINT</li> <li>SQL_C_SHORT</li> <li>SQL_C_TYPE_DATE</li> <li>SQL_C_TYPE_TIME</li> <li>SQL_C_TYPE_TIMESTAMP</li> <li>SQL_C_TYPE_TIMESTAMP_EXT</li> <li>SQL_C_TINYINT</li> <li>SQL_C_UBIGINT</li> <li>SQL_C_UTINYINT</li> <li>SQL_C_WCHAR</li> </ul> <p>Specifying SQL_ARD_TYPE results in the data being converted to the data type specified in the SQL_DESC_CONCISE_TYPE field of the ARD.</p> <p>Specifying SQL_C_DEFAULT results in the data being converted to its default C data type.</p>
SQLPOINTER	<i>TargetValuePtr</i>	output	Pointer to buffer where the retrieved column data is to be stored.
SQLLEN	<i>BufferLength</i>	input	Maximum size of the buffer pointed to by <i>TargetValuePtr</i> . This value is ignored when the driver returns fixed-length data.

## SQLGetData function (CLI) - Get data from a column

Table 81. SQLGetData arguments (continued)

Data type	Argument	Use	Description
SQLLEN *	<i>StrLen_or_IndPtr</i>	output	<p>Pointer to value which indicates the number of bytes CLI has available to return in the <i>TargetValuePtr</i> buffer. If the data is being retrieved in pieces, this contains the number of bytes still remaining.</p> <p>The value is SQL_NULL_DATA if the data value of the column is null. If this pointer is NULL and SQLFetch() has obtained a column containing null data, then this function will fail because it has no means of reporting this.</p> <p>If SQLFetch() has fetched a column containing binary data, then the pointer to <i>StrLen_or_IndPtr</i> must not be NULL or this function will fail because it has no other means of informing the application about the length of the data retrieved in the <i>TargetValuePtr</i> buffer.</p>

**Note:** CLI will provide some performance enhancement if *TargetValuePtr* is placed consecutively in memory after *StrLen\_or\_IndPtr*

### Usage

Different DB2 data sources have different restrictions on how SQLGetData() can be used. For an application to be sure about the functional capabilities of this function, it should call SQLGetInfo() with any of the following SQL\_GETDATA\_EXTENSIONS options:

- SQL\_GD\_ANY\_COLUMN: If this option is returned, SQLGetData() can be called for any unbound column, including those before the last bound column. All DB2 data sources support this feature.
- SQL\_GD\_ANY\_ORDER: If this option is returned, SQLGetData() can be called for unbound columns in any order. All DB2 data sources support this feature.
- SQL\_GD\_BLOCK: If this option is returned by SQLGetInfo() for the SQL\_GETDATA\_EXTENSIONS *InfoType* argument, then the driver will support calls to SQLGetData() when the rowset size is greater than 1. The application can also call SQLSetPos() with the SQL\_POSITION option to position the cursor on the correct row before calling SQLGetData(). At least DB2 for UNIX and Windows data sources support this feature.
- SQL\_GD\_BOUND: If this option is returned, SQLGetData() can be called for bound columns as well as unbound columns. DB2 Database for Linux, UNIX, and Windows does not currently support this feature.

SQLGetData() can also be used to retrieve long columns if the C data type (*TargetType*) is SQL\_C\_CHAR, SQL\_C\_BINARY, SQL\_C\_DBCHAR, SQL\_C\_WCHAR, or if *TargetType* is SQL\_C\_DEFAULT and the column type denotes a binary or character string.

Upon each SQLGetData() call, if the data available for return is greater than or equal to *BufferLength*, truncation occurs. Truncation is indicated by a function return code of SQL\_SUCCESS\_WITH\_INFO coupled with a SQLSTATE denoting data truncation. The application can call SQLGetData() again, with the same ColumnNumber value, to get subsequent data from the same unbound column starting at the point of truncation. To obtain the entire column, the application

## SQLGetData function (CLI) - Get data from a column

repeats such calls until the function returns `SQL_SUCCESS`. The next call to `SQLGetData()` returns `SQL_NO_DATA_FOUND`.

When the application calls the function `SQLGetData()` to retrieve the actual LOB data it will, by default, make one request to the server and will store the entire LOB in memory provided *BufferLength* is large enough. If *BufferLength* is not large enough to hold the requested LOB data, it will be retrieved in pieces.

Although `SQLGetData()` can be used for the sequential retrieval of LOB column data, use the CLI LOB functions if only a portion of the LOB data or a few sections of the LOB column data are needed:

1. Bind the column to a LOB locator.
2. Fetch the row.
3. Use the locator in a `SQLGetSubString()` call, to retrieve the data in pieces (`SQLGetLength()` and `SQLGetPosition()` might also be required in order to determine the values of some of the arguments).
4. Repeat step 2.

Truncation is also affected by the `SQL_ATTR_MAX_LENGTH` statement attribute. The application can specify that truncation is not to be reported by calling `SQLSetStmtAttr()` with `SQL_ATTR_MAX_LENGTH` and a value for the maximum length to return for any one column, and by allocating a *TargetValuePtr* buffer of the same size (plus the null-terminator). If the column data is larger than the set maximum length, `SQL_SUCCESS` will be returned and the maximum length, not the actual length will be returned in *StrLen\_or\_IndPtr*.

To discard the column data part way through the retrieval, the application can call `SQLGetData()` with *ColumnNumber* set to the next column position of interest. To discard data that has not been retrieved for the entire row, the application should call `SQLFetch()` to advance the cursor to the next row; or, if it does not want any more data from the result set, the application can close the cursor by calling `SQLCloseCursor()` or `SQLFreeStmt()` with the `SQL_CLOSE` or `SQL_DROP` option.

The *TargetType* input argument determines the type of data conversion (if any) needed before the column data is placed into the storage area pointed to by *TargetValuePtr*.

For SQL graphic column data:

- The length of the *TargetValuePtr* buffer (*BufferLength*) should be a multiple of 2. The application can determine the SQL data type of the column by first calling `SQLDescribeCol()` or `SQLColAttribute()`.
- The pointer to *StrLen\_or\_IndPtr* must not be NULL since CLI will be storing the number of octets stored in *TargetValuePtr*.
- If the data is to be retrieved in piecewise fashion, CLI will attempt to fill *TargetValuePtr* to the nearest multiple of two octets that is still less than or equal to *BufferLength*. This means if *BufferLength* is not a multiple of two, the last byte in that buffer will be untouched; CLI will not split a double-byte character.

The content returned in *TargetValuePtr* is always null-terminated unless the column data to be retrieved is binary, or if the SQL data type of the column is graphic (DBCS) and the C buffer type is `SQL_C_CHAR`. If the application is retrieving the data in multiple chunks, it should make the proper adjustments (for example, strip off the null-terminator before concatenating the pieces back together assuming the null termination environment attribute is in effect).

## SQLGetData function (CLI) - Get data from a column

Truncation of numeric data types is reported as a warning if the truncation involves digits to the right of the decimal point. If truncation occurs to the left of the decimal point, an error is returned (refer to the diagnostics section).

With the exception of scrollable cursors, applications that use `SQLFetchScroll()` to retrieve data should call `SQLGetData()` only when the rowset size is 1 (equivalent to issuing `SQLFetch()`). `SQLGetData()` can only retrieve column data for a row where the cursor is currently positioned.

### Using SQLGetData() with Scrollable Cursors

`SQLGetData()` can also be used with scrollable cursors. You can save a pointer to any row in the result set with a bookmark. The application can then use that bookmark as a relative position to retrieve a rowset of information.

Once you have positioned the cursor to a row in a rowset using `SQLSetPos()`, you can obtain the bookmark value from column 0 using `SQLGetData()`. In most cases you will not want to bind column 0 and retrieve the bookmark value for every row, but use `SQLGetData()` to retrieve the bookmark value for the specific row you require.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_STILL_EXECUTING`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`
- `SQL_NO_TOTAL`

`SQL_NO_DATA_FOUND` is returned when the preceding `SQLGetData()` call has retrieved all of the data for this column.

`SQL_SUCCESS` is returned if a zero-length string is retrieved by `SQLGetData()`. If this is the case, `StrLen_or_IndPtr` will contain 0, and `TargetValuePtr` will contain a null terminator.

`SQL_NO_TOTAL` is returned as the length when truncation occurs if the CLI configuration keyword `StreamGetData` is set to 1 and CLI cannot determine the number of bytes still available to return in the output buffer.

If the preceding call to `SQLFetch()` failed, `SQLGetData()` should not be called since the result is undefined.

### Diagnostics

Table 82. `SQLGetData` `SQLSTATEs`

SQLSTATE	Description	Explanation
01004	Data truncated.	Data returned for the specified column ( <code>ColumnNumber</code> ) was truncated. String or numeric values are right truncated. <code>SQL_SUCCESS_WITH_INFO</code> is returned.
07006	Invalid conversion.	The data value cannot be converted to the C data type specified by the argument <code>TargetType</code> .  The function has been called before for the same <code>ColumnNumber</code> value but with a different <code>TargetType</code> value.

## SQLGetData function (CLI) - Get data from a column

Table 82. SQLGetData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07009	Invalid descriptor index.	The value specified for <i>ColumnNumber</i> was equal to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF. The value specified for the argument <i>ColumnNumber</i> was greater than the number of columns in the result set.
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument <i>StrLen_or_IndPtr</i> was a null pointer and the value of the column is null. There is no means to report SQL_NULL_DATA.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for the column would have caused the whole part of the number to be truncated.
22005	Error in assignment.	A returned value was incompatible with the data type denoted by the argument <i>TargetType</i> .
22007	Invalid datetime format.	Conversion from character a string to a datetime format was indicated, but an invalid string representation or value was specified, or the value was an invalid date.
22008	Datetime field overflow.	Datetime field overflow occurred; for example, an arithmetic operation on a date or timestamp has a result that is not within the valid range of dates, or a datetime value cannot be assigned to a bound variable because it is too small.
24000	Invalid cursor state.	The previous SQLFetch() resulted in SQL_ERROR or SQL_NO_DATA found; as a result, the cursor is not positioned on a row.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	<i>TargetType</i> was not a valid data type or SQL_C_DEFAULT.
HY010	Function sequence error.	The specified <i>StatementHandle</i> was not in a cursor positioned state. The function was called without first calling SQLFetch().  The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.  An asynchronously executing function (not this one) was called For the <i>StatementHandle</i> and was still executing when this function was called.  The function was called before a statement was prepared on the statement handle.
HY011	Operation invalid at this time.	Calls to SQLGetData() for previously accessed LOB columns are not allowed. Refer to "AllowGetDataLOBReaccess CLI/ODBC configuration keyword" on page 325 for more information.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

## SQLGetData function (CLI) - Get data from a column

Table 82. SQLGetData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The value of the argument <i>BufferLength</i> is less than 0 and the argument <i>TargetType</i> is SQL_C_CHAR, SQL_C_BINARY, SQL_C_DBCHAR or (SQL_C_DEFAULT and the default type is one of SQL_C_CHAR, SQL_C_BINARY, or SQL_C_DBCHAR).
HYC00	Driver not capable.	The SQL data type for the specified data type is recognized but not supported by CLI.  The requested conversion from the SQL data type to the application data <i>TargetType</i> cannot be performed by CLI or the data source.  The column was bound using SQLBindFileToCol().
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* use SQLGetData to get the results */
/* get data from column 1 */
cliRC = SQLGetData(hstmt,
                  1,
                  SQL_C_SHORT,
                  &deptnumb.val,
                  0,
                  &deptnumb.ind);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* get data from column 2 */
cliRC = SQLGetData(hstmt,
                  2,
                  SQL_C_CHAR,
                  location.val,
                  15,
                  &location.ind);
```

## SQLGetDescField function (CLI) - Get single field settings of descriptor record

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLGetDescField() returns the current settings of a single field of a descriptor record.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetDescFieldW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

## SQLGetDescField function (CLI) - Get single field settings of descriptor record

### Syntax

```
SQLRETURN SQLGetDescField (
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT   RecNumber,
    SQLSMALLINT   FieldIdentifier,
    SQLPOINTER    ValuePtr,          /* Value */
    SQLINTEGER    BufferLength,
    SQLINTEGER    *StringLengthPtr); /* *StringLength */
```

### Function arguments

Table 83. SQLGetDescField arguments

Data type	Argument	Use	Description
SQLHDESC	<i>DescriptorHandle</i>	input	Descriptor handle.
SQLSMALLINT	<i>RecNumber</i>	input	Indicates the descriptor record from which the application seeks information. Descriptor records are numbered from 0, with record number 0 being the bookmark record. If the <i>FieldIdentifier</i> argument indicates a field of the descriptor header record, <i>RecNumber</i> must be 0. If <i>RecNumber</i> is less than <code>SQL_DESC_COUNT</code> , but the row does not contain data for a column or parameter, a call to <code>SQLGetDescField()</code> will return the default values of the fields.
SQLSMALLINT	<i>FieldIdentifier</i>	input	Indicates the field of the descriptor whose value is to be returned.
SQLPOINTER	<i>ValuePtr</i>	output	Pointer to a buffer in which to return the descriptor information. The data type depends on the value of <i>FieldIdentifier</i> .
SQLINTEGER	<i>BufferLength</i>	input	<ul style="list-style-type: none"> <li>If <i>ValuePtr</i> points to a character string, this argument should be the length of <i>*ValuePtr</i>.</li> <li>If <i>ValuePtr</i> is a pointer, but not to a string, then <i>BufferLength</i> should have the value <code>SQL_IS_POINTER</code>.</li> <li>If the value in <i>*ValuePtr</i> is of a Unicode data type the <i>BufferLength</i> argument must be an even number.</li> </ul>
SQLSMALLINT *	<i>StringLengthPtr</i>	output	Pointer to the total number of bytes (excluding the number of bytes required for the null termination character) available to return in <i>*ValuePtr</i> .

### Usage

An application can call `SQLGetDescField()` to return the value of a single field of a descriptor record. A call to `SQLGetDescField()` can return the setting of any field in any descriptor type, including header fields, record fields, and bookmark fields. An application can obtain the settings of multiple fields in the same or different descriptors, in arbitrary order, by making repeated calls to `SQLGetDescField()`. `SQLGetDescField()` can also be called to return CLI defined descriptor fields.

For performance reasons, an application should not call `SQLGetDescField()` for an IRD before executing a statement. Calling `SQLGetDescField()` in this case causes the CLI driver to describe the statement, resulting in an extra network flow. When deferred prepare is on and `SQLGetDescField()` is called, you lose the benefit of deferred prepare because the statement must be prepared at the server to obtain describe information.

## SQLGetDescField function (CLI) - Get single field settings of descriptor record

The settings of multiple fields that describe the name, data type, and storage of column or parameter data can also be retrieved in a single call to `SQLGetDescRec()`. `SQLGetStmtAttr()` can be called to return the value of a single field in the descriptor header that has an associated statement attribute.

When an application calls `SQLGetDescField()` to retrieve the value of a field that is undefined for a particular descriptor type, the function returns `SQLSTATE HY091` (Invalid descriptor field identifier). When an application calls `SQLGetDescField()` to retrieve the value of a field that is defined for a particular descriptor type, but has no default value and has not been set yet, the function returns `SQL_SUCCESS` but the value returned for the field is undefined. Refer to the list of initialization values of descriptor fields for any default values which may exist.

The `SQL_DESC_ALLOC_TYPE` header field is available as read-only. This field is defined for all types of descriptors.

Each of these fields is defined either for the IRD only, or for both the IRD and the IPD.

<code>SQL_DESC_AUTO_UNIQUE_VALUE</code>	<code>SQL_DESC_LITERAL_SUFFIX</code>
<code>SQL_DESC_BASE_COLUMN_NAME</code>	<code>SQL_DESC_LOCAL_TYPE_NAME</code>
<code>SQL_DESC_CASE_SENSITIVE</code>	<code>SQL_DESC_SCHEMA_NAME</code>
<code>SQL_DESC_CATALOG_NAME</code>	<code>SQL_DESC_SEARCHABLE</code>
<code>SQL_DESC_DISPLAY_SIZE</code>	<code>SQL_DESC_TABLE_NAME</code>
<code>SQL_DESC_FIXED_PREC_SCALE</code>	<code>SQL_DESC_TYPE_NAME</code>
<code>SQL_DESC_LABEL</code>	<code>SQL_DESC_UNSIGNED</code>
<code>SQL_DESC_LITERAL_PREFIX</code>	<code>SQL_DESC_UPDATABLE</code>

Refer to the list of descriptor *FieldIdentifier* values for more information about the above fields.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_NO_DATA`
- `SQL_INVALID_HANDLE`

`SQL_NO_DATA` is returned if *RecNumber* is greater than the number of descriptor records.

`SQL_NO_DATA` is returned if *DescriptorHandle* is an IRD handle and the statement is in the prepared or executed state, but there was no open cursor associated with it.

### Diagnostics

Table 84. *SQLGetDescField* SQLSTATES

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated.	The buffer <i>*ValuePtr</i> was not large enough to return the entire descriptor field, so the field was truncated. The length of the untruncated descriptor field is returned in <i>*StringLengthPtr</i> . (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)



## SQLGetDescField function (CLI) - Get single field settings of descriptor record

Table 84. SQLGetDescField SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07009	Invalid descriptor index.	<p>The value specified for the <i>RecNumber</i> argument was less than 1, the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF, and the field was not a header field or a CLI defined field.</p> <p>The <i>FieldIdentifier</i> argument was a record field, and the <i>RecNumber</i> argument was 0.</p> <p>The <i>RecNumber</i> argument was less than 0, and the field was not a header field or a CLI defined field.</p>
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY007	Associated statement is not prepared.	<i>DescriptorHandle</i> was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error.	<p><i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called.</p> <p><i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which SQLExecute() or SQLExecDirect() was called and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY021	Inconsistent descriptor information.	The descriptor information checked during a consistency check was not consistent.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
HY091	Invalid descriptor field identifier.	<p><i>FieldIdentifier</i> was undefined for the <i>DescriptorHandle</i>.</p> <p>The value specified for the <i>RecNumber</i> argument was greater than the value in the SQL_DESC_COUNT field.</p>

### Restrictions

None.

### Example

```
/* see how the field SQL_DESC_PARAMETER_TYPE is set */
cliRC = SQLGetDescField(hIPD,
                        1, /* look at the parameter */
                        SQL_DESC_PARAMETER_TYPE,
                        &descFieldParameterType, /* result */
                        SQL_IS_SMALLINT,
```

## SQLGetDescField function (CLI) - Get single field settings of descriptor record

```

NULL);

/* ... */

/* see how the descriptor record field SQL_DESC_TYPE_NAME is set */
rc = SQLGetDescField(hIRD,
    (SQLSMALLINT)colCount,
    SQL_DESC_TYPE_NAME, /* record field */
    descFieldType, /* result */
    25,
    NULL);

/* ... */

/* see how the descriptor record field SQL_DESC_LABEL is set */
rc = SQLGetDescField(hIRD,
    (SQLSMALLINT)colCount,
    SQL_DESC_LABEL, /* record field */
    descFieldLabel, /* result */
    25,
    NULL);

```

---

## SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLGetDescRec() returns the current settings of multiple fields of a descriptor record. The fields returned describe the name, data type, and storage of column or parameter data.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetDescRecW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```

SQLRETURN SQLGetDescRec (
    SQLHDESC          DescriptorHandle, /* hDesc */
    SQLSMALLINT       RecNumber,
    SQLCHAR           *Name,
    SQLSMALLINT       BufferLength,
    SQLSMALLINT       *StringLengthPtr, /* *StringLength */
    SQLSMALLINT       *TypePtr,      /* *Type */
    SQLSMALLINT       *SubTypePtr,   /* *SubType */
    SQLLEN            *LengthPtr,    /* *Length */
    SQLSMALLINT       *PrecisionPtr, /* *Precision */
    SQLSMALLINT       *ScalePtr,    /* *Scale */
    SQLSMALLINT       *NullablePtr); /* *Nullable */

```

### Function arguments

Table 85. SQLGetDescRec arguments

Data type	Argument	Use	Description
SQLHDESC	DescriptorHandle	input	Descriptor handle.

## SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record

Table 85. SQLGetDescRec arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>RecNumber</i>	input	Indicates the descriptor record from which the application seeks information. Descriptor records are numbered from 0, with record number 0 being the bookmark record. The <i>RecNumber</i> argument must be less than or equal to the value of SQL_DESC_COUNT. If <i>RecNumber</i> is less than SQL_DESC_COUNT, but the row does not contain data for a column or parameter, a call to SQLGetDescRec() will return the default values of the fields.
SQLCHAR *	<i>Name</i>	output	A pointer to a buffer in which to return the SQL_DESC_NAME field for the descriptor record.
SQLINTEGER	<i>BufferLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>*Name</i> buffer.
SQLSMALLINT *	<i>StringLengthPtr</i>	output	A pointer to a buffer in which to return the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) available to return in the <i>Name</i> buffer, excluding the null-termination character. If the number of SQLCHAR or SQLWCHAR elements was greater than or equal to <i>BufferLength</i> , the data in <i>*Name</i> is truncated to <i>BufferLength</i> minus the length of a null-termination character, and is null terminated by CLI.
SQLSMALLINT *	<i>TypePtr</i>	output	A pointer to a buffer in which to return the value of the SQL_DESC_TYPE field for the descriptor record.
SQLSMALLINT *	<i>SubTypePtr</i>	output	For records whose type is SQL_DATETIME, this is a pointer to a buffer in which to return the value of the SQL_DESC_DATETIME_INTERVAL_CODE field.
SQLLEN *	<i>LengthPtr</i>	output	A pointer to a buffer in which to return the value of the SQL_DESC_OCTET_LENGTH field for the descriptor record.
SQLSMALLINT *	<i>PrecisionPtr</i>	output	A pointer to a buffer in which to return the value of the SQL_DESC_PRECISION field for the descriptor record.
SQLSMALLINT *	<i>ScalePtr</i>	output	A pointer to a buffer in which to return the value of the SQL_DESC_SCALE field for the descriptor record.
SQLSMALLINT *	<i>NullablePtr</i>	output	A pointer to a buffer in which to return the value of the SQL_DESC_NULLABLE field for the descriptor record.

### Usage

An application can call SQLGetDescRec() to retrieve the values of the following fields for a single column or parameter:

- SQL\_DESC\_NAME
- SQL\_DESC\_TYPE
- SQL\_DESC\_DATETIME\_INTERVAL\_CODE (for records whose type is SQL\_DATETIME)
- SQL\_DESC\_OCTET\_LENGTH

## SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record

- SQL\_DESC\_PRECISION
- SQL\_DESC\_SCALE
- SQL\_DESC\_NULLABLE

SQLGetDescRec() does not retrieve the values for header fields.

An application can inhibit the return of a field's setting by setting the argument corresponding to the field to a null pointer. When an application calls SQLGetDescRec() to retrieve the value of a field that is undefined for a particular descriptor type, the function returns SQL\_SUCCESS but the value returned for the field is undefined. For example, calling SQLGetDescRec() for the SQL\_DESC\_NAME or SQL\_DESC\_NULLABLE field of an APD or ARD will return SQL\_SUCCESS but an undefined value for the field.

When an application calls SQLGetDescRec() to retrieve the value of a field that is defined for a particular descriptor type, but has no default value and has not been set yet, the function returns SQL\_SUCCESS but the value returned for the field is undefined.

The values of fields can also be retrieved individually by a call to SQLGetDescField().

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_NO\_DATA
- SQL\_INVALID\_HANDLE

SQL\_NO\_DATA is returned if *RecNumber* is greater than the number of descriptor records.

SQL\_NO\_DATA is returned if *DescriptorHandle* is an IRD handle and the statement in the prepared or executed state, but there was no open cursor associated with it.

### Diagnostics

Table 86. SQLGetDescRec SQLSTATES

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The buffer <i>*Name</i> was not large enough to return the entire descriptor field, so the field was truncated. The length of the untruncated descriptor field is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index.	The <i>RecNumber</i> argument was set to 0 and the <i>DescriptorHandle</i> argument was an IPD handle.  The <i>RecNumber</i> argument was set to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.  The <i>RecNumber</i> argument was less than 0.
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.

## SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record

Table 86. SQLGetDescRec SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY007	Associated statement is not prepared.	<i>DescriptorHandle</i> was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error.	<i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called.  <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which SQLExecute() or SQLExecDirect() was called and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

### Restrictions

None.

### Example

```
/* get multiple field settings of descriptor record */
rc = SQLGetDescRec(hIRD,
                  i,
                  colname,
                  sizeof(colname),
                  &namelen,
                  &type,
                  &subtype,
                  &width,
                  &precision,
                  &scale,
                  &nullable);

/* ... */

/* get the record/column value after setting */
rc = SQLGetDescRec(hARD,
                  i,
                  colname,
                  sizeof(colname),
                  &namelen,
                  &type,
                  &subtype,
                  &width,
                  &precision,
                  &scale,
                  &nullable);
```

## SQLGetDiagField function (CLI) - Get a field of diagnostic data

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLGetDiagField() returns the current value of a field of a diagnostic data structure, associated with a specific handle, that contains error, warning, and status information.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetDiagFieldW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLGetDiagField (
    SQLSMALLINT HandleType, /* fHandleType */
    SQLHANDLE Handle, /* hHandle */
    SQLSMALLINT RecNumber, /* iRecNumber */
    SQLSMALLINT DiagIdentifier, /* fDiagIdentifier */
    SQLPOINTER DiagInfoPtr, /* pDiagInfo */
    SQLSMALLINT BufferLength, /* cbDiagInfoMax */
    SQLSMALLINT *StringLengthPtr); /* *pcgDiagInfo */
```

### Function arguments

Table 87. SQLGetDiagField arguments

Data type	Argument	Use	Description
SQLSMALLINT	<i>HandleType</i>	input	A handle type identifier that describes the type of handle for which diagnostics are desired. Must be one of the following: <ul style="list-style-type: none"> <li>SQL_HANDLE_ENV</li> <li>SQL_HANDLE_DBC</li> <li>SQL_HANDLE_STMT</li> <li>SQL_HANDLE_DESC</li> </ul>
SQLHANDLE	<i>Handle</i>	input	A handle for the diagnostic data structure, of the type indicated by <i>HandleType</i> .
SQLSMALLINT	<i>RecNumber</i>	input	Indicates the status record from which the application seeks information. Status records are numbered from 1. If the <i>DiagIdentifier</i> argument indicates any field of the diagnostics header record, <i>RecNumber</i> must be 0. If not, it should be greater than 0.
SQLSMALLINT	<i>DiagIdentifier</i>	input	Indicates the field of the diagnostic data structure whose value is to be returned. For more information, see <i>DiagIdentifier</i> argument.
SQLPOINTER	<i>DiagInfoPtr</i>	output	Pointer to a buffer in which to return the diagnostic information. The data type depends on the value of <i>DiagIdentifier</i> .

## SQLGetDiagField function (CLI) - Get a field of diagnostic data

Table 87. SQLGetDiagField arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>BufferLength</i>	input	<p>If <i>DiagIdentifier</i> is ODBC-defined diagnostic:</p> <ul style="list-style-type: none"> <li>If <i>DiagInfoPtr</i> points to a character string or binary buffer, <i>BufferLength</i> should be the length of <i>*DiagInfoPtr</i>.</li> <li>If <i>*DiagInfoPtr</i> is an integer, <i>BufferLength</i> is ignored.</li> <li>If <i>*DiagInfoPtr</i> is a Unicode string, <i>BufferLength</i> must be an even number.</li> </ul> <p>If <i>DiagIdentifier</i> is a CLI diagnostic:</p> <ul style="list-style-type: none"> <li>If <i>*DiagInfoPtr</i> is a pointer to a character string, <i>BufferLength</i> is the number of bytes needed to store the string, or SQL_NTS.</li> <li>If <i>*DiagInfoPtr</i> is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in <i>BufferLength</i>. This places a negative value in <i>BufferLength</i>.</li> <li>If <i>*DiagInfoPtr</i> is a pointer to a value other than a character string or binary string, then <i>BufferLength</i> should have the value SQL_IS_POINTER.</li> <li>If <i>*DiagInfoPtr</i> contains a fixed-length data type, then <i>BufferLength</i> is SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.</li> </ul>
SQLSMALLINT *	<i>StringLengthPtr</i>	output	<p>Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the number of bytes required for the null-termination character, available to return in <i>*DiagInfoPtr</i>, for character data. If the number of bytes available to return is greater than <i>BufferLength</i>, then the text in <i>*DiagInfoPtr</i> is truncated to <i>BufferLength</i> minus the length of a null-termination character. This argument is ignored for non-character data.</p>

### Usage

An application typically calls SQLGetDiagField() to accomplish one of three goals:

1. To obtain specific error or warning information when a function call has returned the SQL\_ERROR or SQL\_SUCCESS\_WITH\_INFO (or SQL\_NEED\_DATA for the SQLBrowseConnect() function) return codes.
2. To find out the number of rows in the data source that were affected when insert, delete, or update operations were performed with a call to SQLExecute(), SQLExecDirect(), SQLBulkOperations(), or SQLSetPos() (from the SQL\_DIAG\_ROW\_COUNT header field), or to find out the number of rows that exist in the current open static scrollable cursor (from the SQL\_DIAG\_CURSOR\_ROW\_COUNT header field).
3. To determine which function was executed by a call to SQLExecDirect() or SQLExecute() (from the SQL\_DIAG\_DYNAMIC\_FUNCTION and SQL\_DIAG\_DYNAMIC\_FUNCTION\_CODE header fields).

## SQLGetDiagField function (CLI) - Get a field of diagnostic data

Any CLI function can post zero or more errors each time it is called, so an application can call `SQLGetDiagField()` after any function call. `SQLGetDiagField()` retrieves only the diagnostic information most recently associated with the diagnostic data structure specified in the *Handle* argument. If the application calls another function, any diagnostic information from a previous call with the same handle is lost.

An application can scan all diagnostic records by incrementing *RecNumber*, as long as `SQLGetDiagField()` returns `SQL_SUCCESS`. The number of status records is indicated in the `SQL_DIAG_NUMBER` header field. Calls to `SQLGetDiagField()` are non-destructive as far as the header and status records are concerned. The application can call `SQLGetDiagField()` again at a later time to retrieve a field from a record, as long as another function other than `SQLGetDiagField()`, `SQLGetDiagRec()`, or `SQLError()` has not been called in the interim, which would post records on the same handle.

An application can call `SQLGetDiagField()` to return any diagnostic field at any time, with the exception of `SQL_DIAG_ROW_COUNT`, which will return `SQL_ERROR` if *Handle* was not a statement handle on which an SQL statement had been executed. If any other diagnostic field is undefined, the call to `SQLGetDiagField()` will return `SQL_SUCCESS` (provided no other error is encountered), and an undefined value is returned for the field.

### HandleType argument

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of *Handle*.

Some header and record fields cannot be returned for all types of handles: environment, connection, statement, and descriptor. Those handles for which a field is not applicable are indicated in the Header Field and Record Fields sections below.

No CLI-specific header diagnostic field should be associated with an environment handle.

### DiagIdentifier argument

This argument indicates the identifier of the field desired from the diagnostic data structure. If *RecNumber* is greater than or equal to 1, the data in the field describes the diagnostic information returned by a function. If *RecNumber* is 0, the field is in the header of the diagnostic data structure, so it contains data pertaining to the function call that returned the diagnostic information, not the specific information. Refer to the list of header and record fields for the *DiagIdentifier* argument for further information.

### Sequence of status records

Status records are placed in a sequence based upon row number and the type of the diagnostic.

If there are two or more status records, the sequence of the records is determined first by row number. The following rules apply to determining the sequence of errors by row:



## SQLGetDiagField function (CLI) - Get a field of diagnostic data

- Records that do not correspond to any row appear in front of records that correspond to a particular row, since SQL\_NO\_ROW\_NUMBER is defined to be -1.
- Records for which the row number is unknown appear in front of all other records, since SQL\_ROW\_NUMBER\_UNKNOWN is defined to be -2.
- For all records that pertain to specific rows, records are sorted by the value in the SQL\_DIAG\_ROW\_NUMBER field. All errors and warnings of the first row affected are listed, then all errors and warnings of the next row affected, and so on.

Within each row, or for all those records that do not correspond to a row or for which the row number is unknown, the first record listed is determined using a set of sorting rules. After the first record, the order of the other records affecting a row is undefined. An application cannot assume that errors precede warnings after the first record. Applications should scan the entire diagnostic data structure to obtain complete information on an unsuccessful call to a function.

The following rules are followed to determine the first record within a row. The record with the highest rank is the first record.

- **Errors.** Status records that describe errors have the highest rank. The following rules are followed to sort errors:
  - Records that indicate a transaction failure or possible transaction failure outrank all other records.
  - If two or more records describe the same error condition, then SQLSTATEs defined by the X/Open CLI specification (classes 03 through HZ) outrank ODBC- and driver-defined SQLSTATEs.
- **Implementation-defined No Data values.** Status records that describe CLI No Data values (class 02) have the second highest rank.
- **Warnings.** Status records that describe warnings (class 01) have the lowest rank. If two or more records describe the same warning condition, then warning SQLSTATEs defined by the X/Open CLI specification outrank ODBC- and driver-defined SQLSTATEs.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA

### Diagnostics

SQLGetDiagField() does not post error values for itself. It uses the following return values to report the outcome of its own execution:

- SQL\_SUCCESS: The function successfully returned diagnostic information.
- SQL\_SUCCESS\_WITH\_INFO: *\*DiagInfoPtr* was too small to hold the requested diagnostic field so the data in the diagnostic field was truncated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to *\*StringLengthPtr*.
- SQL\_INVALID\_HANDLE: The handle indicated by *HandleType* and *Handle* was not a valid handle.
- SQL\_ERROR: One of the following occurred:

## SQLGetDiagField function (CLI) - Get a field of diagnostic data

- The *DiagIdentifier* argument was not one of the valid values.
- The *DiagIdentifier* argument was SQL\_DIAG\_CURSOR\_ROW\_COUNT, SQL\_DIAG\_DYNAMIC\_FUNCTION, SQL\_DIAG\_DYNAMIC\_FUNCTION\_CODE, or SQL\_DIAG\_ROW\_COUNT, but *Handle* was not a statement handle.
- The *RecNumber* argument was negative or 0 when *DiagIdentifier* indicated a field from a diagnostic record. *RecNumber* is ignored for header fields.
- The value requested was a character string and *BufferLength* was less than zero.
- SQL\_NO\_DATA: *RecNumber* was greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL\_NO\_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

### Restrictions

None.

---

## SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLGetDiagRec() returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information. Unlike SQLGetDiagField(), which returns one diagnostic field per call, SQLGetDiagRec() returns several commonly used fields of a diagnostic record: the SQLSTATE, native error code, and error message text.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetDiagRecW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLGetDiagRec (
    SQLSMALLINT HandleType, /* fHandleType */
    SQLHANDLE   Handle,     /* hHandle */
    SQLSMALLINT RecNumber,  /* iRecNumber */
    SQLCHAR     *SQLState,  /* *pszSqlState */
    SQLINTEGER  *NativeErrorPtr, /* *pfNativeError */
    SQLCHAR     *MessageText, /* *pszErrorMsg */
    SQLSMALLINT BufferLength, /* cbErrorMsgMax */
    SQLSMALLINT *TextLengthPtr); /* *pcbErrorMsg */
```

## SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record

### Function arguments

Table 88. SQLGetDiagRec arguments

Data type	Argument	Use	Description
SQLSMALLINT	<i>HandleType</i>	input	A handle type identifier that describes the type of handle for which diagnostics are desired. Must be one of the following: <ul style="list-style-type: none"><li>• SQL_HANDLE_ENV</li><li>• SQL_HANDLE_DBC</li><li>• SQL_HANDLE_STMT</li><li>• SQL_HANDLE_DESC</li></ul>
SQLHANDLE	<i>Handle</i>	input	A handle for the diagnostic data structure, of the type indicated by <i>HandleType</i> .
SQLSMALLINT	<i>RecNumber</i>	input	Indicates the status record from which the application seeks information. Status records are numbered from 1.
SQLCHAR *	<i>SQLState</i>	output	Pointer to a buffer in which to return 5 characters plus a NULL terminator for the SQLSTATE code pertaining to the diagnostic record <i>RecNumber</i> . The first two characters indicate the class; the next three indicate the subclass.
SQLINTEGER *	<i>NativeErrorPtr</i>	output	Pointer to a buffer in which to return the native error code, specific to the data source.
SQLCHAR *	<i>MessageText</i>	output	Pointer to a buffer in which to return the error message text. The fields returned by SQLGetDiagRec() are contained in a text string.
SQLINTEGER	<i>BufferLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>MessageText</i> buffer.
SQLSMALLINT *	<i>TextLengthPtr</i>	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in <i>*MessageText</i> . If the number of SQLCHAR or SQLWCHAR elements available to return is greater than <i>BufferLength</i> , then the error message text in <i>*MessageText</i> is truncated to <i>BufferLength</i> minus the length of a null-termination character.

### Usage

An application typically calls SQLGetDiagRec() when a previous call to a CLI function has returned anything other than SQL\_SUCCESS. However, any function can post zero or more errors each time it is called, so an application can call SQLGetDiagRec() after any function call. An application can call SQLGetDiagRec() multiple times to return some or all of the records in the diagnostic data structure.

SQLGetDiagRec() returns a character string containing the following fields of the diagnostic data structure record:

#### **SQL\_DIAG\_MESSAGE\_TEXT (return type CHAR \*)**

An informational message on the error or warning.

#### **SQL\_DIAG\_NATIVE (return type SQLINTEGER)**

A driver/data-source-specific native error code. If there is no native error code, the driver returns 0.

## SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record

### SQL\_DIAG\_SQLSTATE (return type CHAR \*)

A five-character SQLSTATE diagnostic code.

SQLGetDiagRec() cannot be used to return fields from the header of the diagnostic data structure (the *RecNumber* argument must be greater than 0). The application should call SQLGetDiagField() for this purpose.

SQLGetDiagRec() retrieves only the diagnostic information most recently associated with the handle specified in the *Handle* argument. If the application calls another function, except SQLGetDiagRec() or SQLGetDiagField(), any diagnostic information from the previous calls on the same handle is lost.

An application can scan all diagnostic records by looping, incrementing *RecNumber*, as long as SQLGetDiagRec() returns SQL\_SUCCESS. Calls to SQLGetDiagRec() are non-destructive to the header and record fields. The application can call SQLGetDiagRec() again at a later time to retrieve a field from a record, as long as no other function, except SQLGetDiagRec() or SQLGetDiagField(), has been called in the interim. The application can call SQLGetDiagField() to retrieve the value of the SQL\_DIAG\_NUMBER field, which is the total number of diagnostic records available. SQLGetDiagRec() should then be called that many times.

### HandleType argument

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of *Handle*.

Some header and record fields cannot be returned for all types of handles: environment, connection, statement, and descriptor. Those handles for which a field is not applicable are indicated in the list of header and record fields for the *DiagIdentifier* argument.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

SQLGetDiagRec() does not post error values for itself. It uses the following return values to report the outcome of its own execution:

- SQL\_SUCCESS: The function successfully returned diagnostic information.
- SQL\_SUCCESS\_WITH\_INFO: The *\*MessageText* buffer was too small to hold the requested diagnostic message. No diagnostic records were generated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to *\*StringLengthPtr*.
- SQL\_INVALID\_HANDLE: The handle indicated by *HandleType* and *Handle* was not a valid handle.
- SQL\_ERROR: One of the following occurred:
  - *RecNumber* was negative or 0.
  - *BufferLength* was less than zero.
- SQL\_NO\_DATA: *RecNumber* was greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL\_NO\_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

## SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record

### Example

```
/* get multiple fields settings of diagnostic record */
SQLGetDiagRec(SQL_HANDLE_STMT,
              hstmt,
              1,
              sqlstate,
              &sqlcode,
              message,
              200,
              &length);
```

---

## SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value

### Purpose

Specification:	CLI 2.1		ISO CLI
----------------	---------	--	---------

SQLGetEnvAttr() returns the current setting for the specified environment attribute.

These options are set using the SQLSetEnvAttr() function.

### Syntax

```
SQLRETURN SQLGetEnvAttr (
    SQLHENV EnvironmentHandle, /* henv */
    SQLINTEGER Attribute,
    SQLPOINTER ValuePtr,      /* Value */
    SQLINTEGER BufferLength,
    SQLINTEGER *StringLengthPtr); /* StringLength */
```

### Function arguments

Table 89. SQLGetEnvAttr arguments

Data type	Argument	Use	Description
SQLHENV	<i>EnvironmentHandle</i>	input	Environment handle.
SQLINTEGER	<i>Attribute</i>	input	Attribute to receive. Refer to the list of environment attributes and their descriptions.
SQLPOINTER	<i>ValuePtr</i>	output	A pointer to memory in which to return the current value of the attribute specified by <i>Attribute</i> .
SQLINTEGER	<i>BufferLength</i>	input	Maximum size of buffer pointed to by <i>ValuePtr</i> , if the attribute value is a character string; otherwise, ignored.
SQLINTEGER *	<i>StringLengthPtr</i>	output	Pointer to a buffer in which to return the total number of bytes (excluding the number of bytes returned for the null-termination character) available to return in <i>ValuePtr</i> . If <i>ValuePtr</i> is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than or equal to <i>BufferLength</i> , the data in <i>ValuePtr</i> is truncated to <i>BufferLength</i> minus the length of a null-termination character and is null-terminated by CLI.

If *Attribute* does not denote a string, then CLI ignores *BufferLength* and does not set *StringLengthPtr*.

## SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value

### Usage

SQLGetEnvAttr() can be called at any time between the allocation and freeing of the environment handle. It obtains the current value of the environment attribute.

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 90. SQLGetEnvAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY092	Option type out of range.	An invalid <i>Attribute</i> value was specified.

### Restrictions

None.

### Example

```
/* retrieve the current environment attribute value */
cliRC = SQLGetEnvAttr(henv, SQL_ATTR_OUTPUT_NTS, &output_nts, 0, NULL);
```

## SQLGetFunctions function (CLI) - Get functions

### Purpose

Specification:	CLI 2.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLGetFunctions() can be used to query whether a specific CLI or ODBC function is supported. This allows applications to adapt to varying levels of support when connecting to different database servers.

A connection to a database server must exist before calling this function.

### Syntax

```
SQLRETURN SQLGetFunctions (
    SQLHDBC          ConnectionHandle, /* hdbc */
    SQLUSMALLINT    FunctionId,      /* fFunction */
    SQLUSMALLINT    *SupportedPtr); /* pfExists */
```

### Function arguments

Table 91. SQLGetFunctions arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Database connection handle.
SQLUSMALLINT	<i>FunctionId</i>	input	The function being queried.

## SQLGetFunctions function (CLI) - Get functions

Table 91. SQLGetFunctions arguments (continued)

Data type	Argument	Use	Description
SQLUSMALLINT *	<i>SupportedPtr</i>	output	Pointer to location where this function will return SQL_TRUE or SQL_FALSE depending on whether the function being queried is supported.

### Usage

If *FunctionId* is set to SQL\_API\_ALL\_FUNCTIONS, then *SupportedPtr* must point to an SQLSMALLINT array of 100 elements. The array is indexed by the *FunctionId* values used to identify many of the functions. Some elements of the array are unused and reserved. Since some *FunctionId* values are greater than 100, the array method can not be used to obtain a list of functions. The SQLGetFunctions() call must be explicitly issued for all *FunctionId* values equal to or above 100. The complete set of *FunctionId* values is defined in sqlcli1.h.

**Note:** The LOB support functions (SQLGetLength(), SQLGetPosition(), SQLGetSubString(), SQLBindFileToCol(), SQLBindFileToCol()) are not supported when connected to IBM RDBMSs that do not support LOB data types.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 92. SQLGetFunctions SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	SQLGetFunctions() was called before a database connection was established.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

### Authorization

None.

### Example

```
/* check to see if SQLGetInfo() is supported */  
cliRC = SQLGetFunctions(hdbc, SQL_API_SQLGETINFO, &supported);
```

## SQLGetFunctions function (CLI) - Get functions

### References

None.

---

## SQLGetInfo function (CLI) - Get general information

Returns general information about the DBMS that the application is currently connected to.

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

The SQLGetInfo() function returns general information about the database management system (DBMS) that the application is currently connected to.

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is SQLGetInfoW(). See “Unicode functions (CLI)” on page 5 for information about ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLGetInfo (
    SQLHDBC          ConnectionHandle, /* hdbc */
    SQLUSMALLINT     InfoType,        /* fInfoType */
    SQLPOINTER       InfoValuePtr,    /* rgbInfoValue */
    SQLSMALLINT      BufferLength,     /* cbInfoValueMax */
    SQLSMALLINT      *StringLengthPtr); /* pcbInfoValue */
```

### Function arguments

Table 93. SQLGetInfo arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	Input	The database connection handle.
SQLUSMALLINT	<i>InfoType</i>	Input	The type of information that is required. The possible values for this argument are described in Information returned by SQLGetInfo().
SQLPOINTER	<i>InfoValuePtr</i>	Output and input	Pointer to buffer where this function stores the information that you want. Depending on the type of information that is being retrieved, 5 types of information can be returned: <ul style="list-style-type: none"><li>• 16-bit integer value</li><li>• 32-bit integer value</li><li>• 32-bit binary value</li><li>• 32-bit mask</li><li>• Null-terminated character string</li></ul> If the <i>InfoType</i> argument is SQL_DRIVER_HDESC or SQL_DRIVER_HSTMT, <i>InfoValuePtr</i> is both input and output argument.
SQLSMALLINT	<i>BufferLength</i>	Input	The maximum length of the buffer pointed by <i>InfoValuePtr</i> pointer. If <i>*InfoValuePtr</i> is a Unicode string, the <i>BufferLength</i> argument must be an even number.



## SQLGetInfo function (CLI) - Get general information

Table 93. SQLGetInfo arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	<i>StringLengthPtr</i>	Output	<p>Pointer to location where this function returns the total number of bytes of information that is available to return. For string output, the length does not include the null terminating character.</p> <p>If the value in the location pointed by <i>StringLengthPtr</i> is greater than the size specified in <i>BufferLength</i>, the string output information would be truncated to <i>BufferLength</i> - 1 bytes and the function returns with SQL_SUCCESS_WITH_INFO.</p>

### Usage

See Information returned by SQLGetInfo() for a list of the possible values of the *InfoType* argument and a description of the information that the SQLGetInfo() function would return for that value.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 94. SQLGetInfo SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated.	The requested information is returned as a string, and its length exceeded the length of the application buffer as specified in the <i>BufferLength</i> argument. The <i>StringLengthPtr</i> argument contains the actual (not truncated) length of the requested information. (Function returns SQL_SUCCESS_WITH_INFO return code.)
08003	Connection is closed.	The type of information that is requested in the <i>InfoType</i> argument requires an open connection. Only the SQL_ODBC_VER information does not require an open connection.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY090	Invalid string or buffer length.	The value specified for the <i>BufferLength</i> argument is less than 0.
HY096	Information type out of range.	An invalid <i>InfoType</i> argument is specified.
HYC00	Driver not capable.	The value specified in the <i>InfoType</i> argument is not supported by either CLI or the data source.

### Restrictions

None.

## SQLGetInfo function (CLI) - Get general information

### Example

```
/* get server name information */
cliRC = SQLGetInfo(hdbc, SQL_DBMS_NAME, imageInfoBuf, 255, &outlen);

/* ... */

/* get client driver name information */
cliRC = SQLGetInfo(hdbc, SQL_DRIVER_NAME, imageInfoBuf, 255, &outlen);
```

### Information that is returned by the SQLGetInfo() function

**Note:** CLI returns a value for each *InfoType* argument in this table. If the *InfoType* argument does not apply or is not supported, the result is dependent on the return type. If the return type is a:

- Character string ("Y" or "N"), "N" is returned.
- Character string (not "Y" or "N"), an empty string is returned.
- 32-bit integer, 0 (zero) is returned.
- 32-bit mask, 0 (zero) is returned.

#### SQL\_ACCESSIBLE\_PROCEDURES (string)

A character string of "Y" indicates that you can run all procedures that are returned by the function `SQLProcedures()`. "N" indicates there might be procedures returned that you cannot run.

#### SQL\_ACCESSIBLE\_TABLES (string)

A character string of "Y" indicates that you are guaranteed SELECT privilege to all tables that are returned by the function `SQLTables()`. "N" indicates that there might be tables returned that you cannot access.

#### SQL\_AGGREGATE\_FUNCTIONS (32-bit mask)

A bit mask that enumerates support for the listed aggregation functions:

- SQL\_AF\_ALL
- SQL\_AF\_AVG
- SQL\_AF\_COUNT
- SQL\_AF\_DISTINCT
- SQL\_AF\_MAX
- SQL\_AF\_MIN
- SQL\_AF\_SUM

#### SQL\_ALTER\_DOMAIN (32-bit mask)

CLI returns 0 that indicates that the ALTER DOMAIN statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_AD\_ADD\_CONSTRAINT\_DEFERRABLE
- SQL\_AD\_ADD\_CONSTRAINT\_NON\_DEFERRABLE
- SQL\_AD\_ADD\_CONSTRAINT\_INITIALLY\_DEFERRED
- SQL\_AD\_ADD\_CONSTRAINT\_INITIALLY\_IMMEDIATE
- SQL\_AD\_ADD\_DOMAIN\_CONSTRAINT
- SQL\_AD\_ADD\_DOMAIN\_DEFAULT
- SQL\_AD\_CONSTRAINT\_NAME\_DEFINITION
- SQL\_AD\_DROP\_DOMAIN\_CONSTRAINT
- SQL\_AD\_DROP\_DOMAIN\_DEFAULT

#### SQL\_ALTER\_TABLE (32-bit mask)

Indicates which clauses in the ALTER TABLE statement are supported by the DBMS:

- SQL\_AT\_ADD\_COLUMN\_COLLATION
- SQL\_AT\_ADD\_COLUMN\_DEFAULT
- SQL\_AT\_ADD\_COLUMN\_SINGLE

## SQLGetInfo function (CLI) - Get general information

- SQL\_AT\_ADD\_CONSTRAINT
- SQL\_AT\_ADD\_TABLE\_CONSTRAINT
- SQL\_AT\_CONSTRAINT\_NAME\_DEFINITION
- SQL\_AT\_DROP\_COLUMN\_CASCADE
- SQL\_AT\_DROP\_COLUMN\_DEFAULT
- SQL\_AT\_DROP\_COLUMN\_RESTRICT
- SQL\_AT\_DROP\_TABLE\_CONSTRAINT\_CASCADE
- SQL\_AT\_DROP\_TABLE\_CONSTRAINT\_RESTRICT
- SQL\_AT\_SET\_COLUMN\_DEFAULT
- SQL\_AT\_CONSTRAINT\_INITIALLY\_DEFERRED
- SQL\_AT\_CONSTRAINT\_INITIALLY\_IMMEDIATE
- SQL\_AT\_CONSTRAINT\_DEFERRABLE
- SQL\_AT\_CONSTRAINT\_NON\_DEFERRABLE

### SQL\_APPLICATION\_CODEPAGE (32-bit unsigned integer)

Indicates the application code page.

### SQL\_ASYNC\_MODE (32-bit unsigned integer)

Indicates the level of asynchronous support in the driver:

- SQL\_AM\_CONNECTION : Connection level asynchronous execution is supported. Either all statement handles that are associated with a given connection handle are in asynchronous mode, or all are in synchronous mode. A statement handle that is on a connection cannot be in asynchronous mode while another statement handle on the same connection is in synchronous mode, and vice versa.
- SQL\_AM\_STATEMENT : Statement level asynchronous execution is supported. Some statement handles that are associated with a connection handle can be in asynchronous mode, while other statement handles on the same connection are in synchronous mode.
- SQL\_AM\_NONE : Asynchronous mode is not supported.

This value is also returned if the CLI/ODBC configuration keyword ASYNCENABLE is set to disable asynchronous execution.

### SQL\_BATCH\_ROW\_COUNT (32-bit mask)

Indicates how row counts are dealt with. CLI always returns SQL\_BRC\_ROLLED\_UP, which indicates that row counts for consecutive INSERT, DELETE, or UPDATE statements are rolled into one.

ODBC also defines the values that are not returned by CLI:

- SQL\_BRC\_PROCEDURES
- SQL\_BRC\_EXPLICIT

### SQL\_BATCH\_SUPPORT (32-bit mask)

Indicates which levels of batches are supported:

- SQL\_BS\_SELECT\_EXPLICIT : Supports explicit batches that can have result-set generating statements.
- SQL\_BS\_ROW\_COUNT\_EXPLICIT : Supports explicit batches that can have row-count generating statements.
- SQL\_BS\_SELECT\_PROC : Supports explicit procedures that can have result-set generating statements.
- SQL\_BS\_ROW\_COUNT\_PROC : Supports explicit procedures that can have row-count generating statements.

### SQL\_BOOKMARK\_PERSISTENCE (32-bit mask)

Indicates when bookmarks remain valid after an operation:

- SQL\_BP\_CLOSE : After an application calls SQLFreeStmt() with the SQL\_CLOSE option, or SQLCloseCursor() to close the cursor associated with a statement.
- SQL\_BP\_DELETE : After that row has been deleted.

## SQLGetInfo function (CLI) - Get general information

- **SQL\_BP\_DROP** : Bookmarks are valid after an application calls `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_STMT` to drop a statement.
- **SQL\_BP\_TRANSACTION** : After an application commits or rolls back a transaction.
- **SQL\_BP\_UPDATE** : After any column in that row has been updated, including key columns.
- **SQL\_BP\_OTHER\_HSTMT** : A bookmark that is associated with one statement can be used with another statement. Unless `SQL_BP_CLOSE` or `SQL_BP_DROP` is specified, the cursor on the first statement must be open.

### **SQL\_CATALOG\_LOCATION (16-bit integer)**

A 16-bit integer value that indicates the position of the qualifier in a qualified table name. CLI always returns `SQL_CL_START` for this information type. ODBC also defines the value `SQL_CL_END` which is not returned by CLI.

In previous versions of CLI this *InfoType* argument was `SQL_QUALIFIER_LOCATION`.

### **SQL\_CATALOG\_NAME (string)**

A character string of "Y" indicates that the server supports catalog names. "N" indicates that catalog names are not supported.

### **SQL\_CATALOG\_NAME\_SEPARATOR (string)**

The character(s) used as a separator between a catalog name and the qualified name element that follows or precedes it.

In previous versions of CLI this *InfoType* was `SQL_QUALIFIER_NAME_SEPARATOR`.

### **SQL\_CATALOG\_TERM (string)**

The qualifier (catalog) terminology that is used by the database vendor.

The name that the vendor uses for the high-order part of a three-part name.

If the target DBMS does not support three-part naming, a zero-length string is returned.

In previous versions of CLI this *InfoType* argument was `SQL_QUALIFIER_TERM`.

### **SQL\_CATALOG\_USAGE (32-bit mask)**

A 32-bit mask enumerating the statements in which you can use catalogs. The `SQL_CATALOG_USAGE` is similar to `SQL_SCHEMA_USAGE`, except that `SQL_CATALOG_USAGE` is specific for catalogs.

- **SQL\_CU\_DML\_STATEMENTS** : All data manipulation language (DML) statements.
- **SQL\_CU\_INDEX\_DEFINITION** : All index definition statements.
- **SQL\_CU\_PRIVILEGE\_DEFINITION** : All privilege definition statements.
- **SQL\_CU\_PROCEDURE\_INVOCATION** : The ODBC procedure invocation statement.
- **SQL\_CU\_TABLE\_DEFINITION** : All table definition statements.

A value of 0 is returned if catalogs are not supported by the data source.

In previous versions of CLI, this *InfoType* argument was `SQL_QUALIFIER_USAGE`.

## SQLGetInfo function (CLI) - Get general information

### SQL\_COLLATION\_SEQ (string)

Indicates the name of the default collation sequence for the default character set for this server (for example ISO 8859-1 or EBCDIC). If the collation sequence is unknown, an empty string is returned.

### SQL\_COLUMN\_ALIAS (string)

Returns "Y" if column aliases are supported, or "N" if they are not.

### SQL\_CONCAT\_NULL\_BEHAVIOR (16-bit integer)

Indicates how the concatenation of NULL valued character data type columns with non-NULL valued character data type columns is handled.

- SQL\_CB\_NULL : A NULL value (this behavior is the case for IBM RDBMS).
- SQL\_CB\_NON\_NULL : A concatenation of non-NULL column values.

### SQL\_CONVERT\_\* (32-bit masks)

SQL\_CONVERT\_BIGINT (32-bit mask)

SQL\_CONVERT\_BINARY (32-bit mask)

SQL\_CONVERT\_BIT (32-bit mask)

SQL\_CONVERT\_CHAR (32-bit mask)

SQL\_CONVERT\_DATE (32-bit mask)

SQL\_CONVERT\_DECIMAL (32-bit mask)

SQL\_CONVERT\_DOUBLE (32-bit mask)

SQL\_CONVERT\_FLOAT (32-bit mask)

SQL\_CONVERT\_INTEGER (32-bit mask)

SQL\_CONVERT\_INTERVAL\_YEAR\_MONTH (32-bit mask)

SQL\_CONVERT\_INTERVAL\_DAY\_TIME (32-bit mask)

SQL\_CONVERT\_LONGVARBINARY (32-bit mask)

SQL\_CONVERT\_LONGVARCHAR (32-bit mask)

SQL\_CONVERT\_NUMERIC (32-bit mask)

SQL\_CONVERT\_REAL (32-bit mask)

SQL\_CONVERT\_SMALLINT (32-bit mask)

SQL\_CONVERT\_TIME (32-bit mask)

SQL\_CONVERT\_TIMESTAMP (32-bit mask)

SQL\_CONVERT\_TINYINT (32-bit mask)

SQL\_CONVERT\_VARBINARY (32-bit mask)

SQL\_CONVERT\_VARCHAR (32-bit mask)

SQL\_CONVERT\_WCHAR (32-bit mask)

SQL\_CONVERT\_WLONGVARCHAR (32-bit mask)

SQL\_CONVERT\_WVARCHAR (32-bit mask)

Indicates the conversions that are supported by the data source with the CONVERT scalar function for data of the type named in the *InfoType*. If the bit mask equals zero, the data source does not support any conversions for the named data type, including conversions to the same data type.

For example, to find out if a data source supports the conversion of SQL\_INTEGER data to the SQL\_DECIMAL data type, an application calls SQLGetInfo() function with the *InfoType* argument of SQL\_CONVERT\_INTEGER. The application then performs AND operation on the returned bit mask with SQL\_CVT\_DECIMAL. If the resulting value is nonzero, the conversion is supported.

The listed bit masks are used to determine which conversions are supported:

- SQL\_CVT\_BIGINT
- SQL\_CVT\_BINARY
- SQL\_CVT\_BIT
- SQL\_CVT\_CHAR

## SQLGetInfo function (CLI) - Get general information

- SQL\_CVT\_DATE
- SQL\_CVT\_DECIMAL
- SQL\_CVT\_DOUBLE
- SQL\_CVT\_FLOAT
- SQL\_CVT\_INTEGER
- SQL\_CVT\_INTERVAL\_YEAR\_MONTH
- SQL\_CVT\_INTERVAL\_DAY\_TIME
- SQL\_CVT\_LONGVARBINARY
- SQL\_CVT\_LONGVARCHAR
- SQL\_CVT\_NUMERIC
- SQL\_CVT\_REAL
- SQL\_CVT\_SMALLINT
- SQL\_CVT\_TIME
- SQL\_CVT\_TIMESTAMP
- SQL\_CVT\_TINYINT
- SQL\_CVT\_VARBINARY
- SQL\_CVT\_VARCHAR
- SQL\_CVT\_WCHAR
- SQL\_CVT\_WLONGVARCHAR
- SQL\_CVT\_WVARCHAR

### SQL\_CONNECT\_CODEPAGE (32-bit unsigned integer)

Indicates the code page of the current connection.

### SQL\_CONVERT\_FUNCTIONS (32-bit mask)

Indicates the scalar conversion functions that are supported by the driver and associated data source.

CLI Version 2.1.1 and later supports ODBC scalar conversions between char variables (CHAR, VARCHAR, LONG VARCHAR, and CLOB) and DOUBLE (or FLOAT).

- SQL\_FN\_CVT\_CONVERT : Used to determine which conversion functions are supported.

### SQL\_CORRELATION\_NAME (16-bit integer)

Indicates the degree of correlation name support by the server:

- SQL\_CN\_ANY : Any valid user-defined name is supported.
- SQL\_CN\_NONE : Correlation name is not supported.
- SQL\_CN\_DIFFERENT : Correlation name is supported, but it must be different than the name of the table that it represents.

### SQL\_CREATE\_ASSERTION (32-bit mask)

Indicates which clauses in the CREATE ASSERTION statement are supported by the DBMS. CLI always returns zero; the CREATE ASSERTION statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_CA\_CREATE\_ASSERTION
- SQL\_CA\_CONSTRAINT\_INITIALLY\_DEFERRED
- SQL\_CA\_CONSTRAINT\_INITIALLY\_IMMEDIATE
- SQL\_CA\_CONSTRAINT\_DEFERRABLE
- SQL\_CA\_CONSTRAINT\_NON\_DEFERRABLE

### SQL\_CREATE\_CHARACTER\_SET (32-bit mask)

Indicates which clauses in the CREATE CHARACTER SET statement are supported by the DBMS. CLI always returns zero; the CREATE CHARACTER SET statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_CCS\_CREATE\_CHARACTER\_SET

## SQLGetInfo function (CLI) - Get general information

- SQL\_CCS\_COLLATE\_CLAUSE
- SQL\_CCS\_LIMITED\_COLLATION

### SQL\_CREATE\_COLLATION (32-bit mask)

Indicates which clauses in the CREATE COLLATION statement are supported by the DBMS. CLI always returns zero; the CREATE COLLATION statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_CCOL\_CREATE\_COLLATION

### SQL\_CREATE\_DOMAIN (32-bit mask)

Indicates which clauses in the CREATE DOMAIN statement are supported by the DBMS. CLI always returns zero; the CREATE DOMAIN statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_CDO\_CREATE\_DOMAIN
- SQL\_CDO\_CONSTRAINT\_NAME\_DEFINITION
- SQL\_CDO\_DEFAULT
- SQL\_CDO\_CONSTRAINT
- SQL\_CDO\_COLLATION
- SQL\_CDO\_CONSTRAINT\_INITIALLY\_DEFERRED
- SQL\_CDO\_CONSTRAINT\_INITIALLY\_IMMEDIATE
- SQL\_CDO\_CONSTRAINT\_DEFERRABLE
- SQL\_CDO\_CONSTRAINT\_NON\_DEFERRABLE

### SQL\_CREATE\_MODULE (32-bit mask)

Indicates which clauses in the CREATE MODULE statement are supported by the DBMS. CLI always returns zero for z/OS.

CLI returns the listed values:

- SQL\_CM\_CREATE\_MODULE
- SQL\_CM\_AUTHORIZATION
- SQL\_CM\_DEFAULT\_CHARACTER\_SET

### SQL\_CREATE\_SCHEMA (32-bit mask)

Indicates which clauses in the CREATE SCHEMA statement are supported by the DBMS:

- SQL\_CS\_CREATE\_SCHEMA
- SQL\_CS\_AUTHORIZATION
- SQL\_CS\_DEFAULT\_CHARACTER\_SET

### SQL\_CREATE\_TABLE (32-bit mask)

Indicates which clauses in the CREATE TABLE statement are supported by the DBMS.

The listed bit masks are used to determine which clauses are supported:

- SQL\_CT\_CREATE\_TABLE
- SQL\_CT\_TABLE\_CONSTRAINT
- SQL\_CT\_CONSTRAINT\_NAME\_DEFINITION

The listed bits specify the ability to create temporary tables:

- SQL\_CT\_COMMIT\_PRESERVE : Deleted rows are preserved on commit.
- SQL\_CT\_COMMIT\_DELETE : Deleted rows are deleted on commit.
- SQL\_CT\_GLOBAL\_TEMPORARY : Global temporary tables can be created.
- SQL\_CT\_LOCAL\_TEMPORARY : Local temporary tables can be created.

The listed bits specify the ability to create column constraints:

## SQLGetInfo function (CLI) - Get general information

- `SQL_CT_COLUMN_CONSTRAINT` : Specifying column constraints is supported.
- `SQL_CT_COLUMN_DEFAULT` : Specifying column defaults is supported.
- `SQL_CT_COLUMN_COLLATION` : Specifying column collation is supported.

The listed bits specify the supported constraint attributes, if specifying column or table constraints is supported:

- `SQL_CT_CONSTRAINT_INITIALLY_DEFERRED`
- `SQL_CT_CONSTRAINT_INITIALLY_IMMEDIATE`
- `SQL_CT_CONSTRAINT_DEFERRABLE`
- `SQL_CT_CONSTRAINT_NON_DEFERRABLE`

### **SQL\_CREATE\_TRANSLATION (32-bit mask)**

Indicates which clauses in the CREATE TRANSLATION statement are supported by the DBMS. CLI always returns zero; the CREATE TRANSLATION statement is not supported.

ODBC also defines the listed value that is not returned by CLI:

- `SQL_CTR_CREATE_TRANSLATION`

### **SQL\_CREATE\_VIEW (32-bit mask)**

Indicates which clauses in the CREATE VIEW statement are supported by the DBMS:

- `SQL_CV_CREATE_VIEW`
- `SQL_CV_CHECK_OPTION`
- `SQL_CV_CASCADED`
- `SQL_CV_LOCAL`

A return value of 0 means that the CREATE VIEW statement is not supported.

### **SQL\_CURSOR\_COMMIT\_BEHAVIOR (16-bit integer)**

Indicates how a COMMIT operation affects cursors. A value of:

- `SQL_CB_DELETE`, deletes cursors and drops access plans for dynamic SQL statements.
- `SQL_CB_CLOSE`, deletes cursors, but retains access plans for dynamic SQL statements (including non-query statements)
- `SQL_CB_PRESERVE`, retains cursors and access plans for dynamic statements (including non-query statements). Applications can continue to fetch data, or close the cursor and re-execute the query without preparing again the statement.

**Note:** After COMMIT, a FETCH must be issued to reposition the cursor before actions such as positioned updates or deletes can be taken.

### **SQL\_CURSOR\_ROLLBACK\_BEHAVIOR (16-bit integer)**

Indicates how a ROLLBACK operation affects cursors. A value of:

- `SQL_CB_DELETE`, deletes cursors and drops access plans for dynamic SQL statements.
- `SQL_CB_CLOSE`, deletes cursors, but retains access plans for dynamic SQL statements (including non-query statements)
- `SQL_CB_PRESERVE`, retains cursors and access plans for dynamic statements (including non-query statements). Applications can continue to fetch data, or close the cursor and re-execute the query without preparing again the statement.

**Note:** DB2 servers do not have the `SQL_CB_PRESERVE` property.



### SQL\_CURSOR\_SENSITIVITY (32-bit unsigned integer)

Indicates support for cursor sensitivity:

- `SQL_INSENSITIVE`, all cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor within the same transaction.
- `SQL_UNSPECIFIED`, it is unspecified whether cursors on the statement handle make visible the changes made to a result set by another cursor within the same transaction. Cursors on the statement handle might make visible none, some, or all such changes.
- `SQL_SENSITIVE`, cursors are sensitive to changes made by other cursors within the same transaction.

### SQL\_DATA\_SOURCE\_NAME (string)

Indicates the data source name used during connection. If the application called `SQLConnect()`, this character string is the value of the `szDSN` argument. If the application called `SQLDriverConnect()` or `SQLBrowseConnect()`, this character string is the value of the DSN keyword in the connection string passed to the driver. If the connection string did not contain the DSN keyword, this character string is an empty string.

### SQL\_DATA\_SOURCE\_READ\_ONLY (string)

A character string of "Y" indicates that the database is set to READ ONLY mode, "N" indicates that is not set to READ ONLY mode. This characteristic pertains only to the data source itself; it is not characteristic of the driver that enables access to the data source.

### SQL\_DATABASE\_CODEPAGE (32-bit unsigned integer)

Indicates the code page of the database that the application is currently connected to.

### SQL\_DATABASE\_NAME (string)

The name of the current database in use

**Note:** This string is the same as that returned by the `SELECT CURRENT SERVER` statement on non-host systems. For host databases, such as DB2 for z/OS or DB2 for i, the string returned is the DCS database name. This database name was provided when the `CATALOG DCS DATABASE DIRECTORY` command was issued at the DB2 Connect gateway.

### SQL\_DATETIME\_LITERALS (32-bit unsigned integer)

Indicates the datetime literals that are supported by the DBMS. CLI always returns zero; datetime literals are not supported.

ODBC also defines the listed values that are not returned by CLI:

- `SQL_DL_SQL92_DATE`
- `SQL_DL_SQL92_TIME`
- `SQL_DL_SQL92_TIMESTAMP`
- `SQL_DL_SQL92_INTERVAL_YEAR`
- `SQL_DL_SQL92_INTERVAL_MONTH`
- `SQL_DL_SQL92_INTERVAL_DAY`
- `SQL_DL_SQL92_INTERVAL_HOUR`
- `SQL_DL_SQL92_INTERVAL_MINUTE`
- `SQL_DL_SQL92_INTERVAL_SECOND`
- `SQL_DL_SQL92_INTERVAL_YEAR_TO_MONTH`
- `SQL_DL_SQL92_INTERVAL_DAY_TO_HOUR`
- `SQL_DL_SQL92_INTERVAL_DAY_TO_MINUTE`
- `SQL_DL_SQL92_INTERVAL_DAY_TO_SECOND`
- `SQL_DL_SQL92_INTERVAL_HOUR_TO_MINUTE`

## SQLGetInfo function (CLI) - Get general information

- SQL\_DL\_SQL92\_INTERVAL\_HOUR\_TO\_SECOND
- SQL\_DL\_SQL92\_INTERVAL\_MINUTE\_TO\_SECOND

### SQL\_DBMS\_NAME (string)

The name of the DBMS product being accessed

For example:

- "DB2/6000"
- "DB2/2"

### SQL\_DBMS\_VER (string)

The Version of the DBMS product accessed. A string of the form 'mm.vv.rrrr' where *mm* is the major version, *vv* is the minor version, and *rrrr* is the release number. For example, "0r.01.0000" translates to major version *r*, minor version 1, release 0.

### SQL\_DDL\_INDEX (32-bit unsigned integer)

Indicates support for the creation and dropping of indexes:

- SQL\_DI\_CREATE\_INDEX
- SQL\_DI\_DROP\_INDEX

### SQL\_DEFAULT\_TXN\_ISOLATION (32-bit mask)

The default transaction isolation level supported

One of the listed masks are returned:

- SQL\_TXN\_READ\_UNCOMMITTED : Changes are immediately perceived by all transactions (dirty read, non-repeatable read, and phantoms are possible).  
This behavior is equivalent to Uncommitted Read level for IBM databases.
- SQL\_TXN\_READ\_COMMITTED : Row read by transaction 1 can be altered and committed by transaction 2 (non-repeatable read and phantoms are possible)  
This behavior is equivalent to Cursor Stability level in IBM databases.
- SQL\_TXN\_REPEATABLE\_READ : A transaction can add or remove rows matching the search condition or a pending transaction (repeatable read, but phantoms are possible)  
This behavior is equivalent to Read Stability level in IBM databases.
- SQL\_TXN\_SERIALIZABLE : Data affected by pending transaction is not available to other transactions (repeatable read, phantoms are not possible)  
This behavior is equivalent to Repeatable Read level in IBM databases.
- SQL\_TXN\_VERSIONING : Not applicable to IBM DBMSs.
- SQL\_TXN\_NOCOMMIT : Any changes are effectively committed at the end of a successful operation; no explicit commit or rollback is allowed.  
This is a IBM DB2 for IBM i isolation level.

In IBM terminology,

- SQL\_TXN\_READ\_UNCOMMITTED is Uncommitted Read;
- SQL\_TXN\_READ\_COMMITTED is Cursor Stability;
- SQL\_TXN\_REPEATABLE\_READ is Read Stability;
- SQL\_TXN\_SERIALIZABLE is Repeatable Read.

### SQL\_DESCRIBE\_PARAMETER (string)

"Y" if parameters can be described; "N" if not.

### SQL\_DM\_VER (string)

Reserved.

## SQLGetInfo function (CLI) - Get general information

### SQL\_DRIVER\_BLDLEVEL

Build level information about the current version of CLI.

The information is in the listed format: sYYMMDD, where YY is the year of the build, MM is the month and DD is the day. For example, **s100610**.

For special builds, the format is: special\_JOBID, where JOBID is the special build's job identification. For example, **special\_39899**.

For full version information, use SQL\_DRIVER\_BLDLEVEL with SQL\_DRIVER\_VER.

### SQL\_DRIVER\_HDBC (32 bits)

CLI's database handle

### SQL\_DRIVER\_HDESC (32 bits)

CLI's descriptor handle

### SQL\_DRIVER\_HENV (32 bits)

CLI's environment handle

### SQL\_DRIVER\_HLIB (32 bits)

Reserved.

### SQL\_DRIVER\_HSTMT (32 bits)

CLI's statement handle

In an ODBC environment with an ODBC Driver Manager, if *InfoType* is set to SQL\_DRIVER\_HSTMT, the Driver Manager statement handle (the one returned from SQLAllocStmt()) must be passed on input in *rgbInfoValue* from the application. In this case *rgbInfoValue* is both an input and an output argument. The ODBC Driver Manager is responsible for returning the mapped value. ODBC applications wishing to call CLI specific functions (such as the LOB functions) can access them, by passing these handle values to the functions after loading the CLI library and issuing an operating system function call to invoke the desired functions.

### SQL\_DRIVER\_NAME (string)

The file name of the CLI implementation.

### SQL\_DRIVER\_ODBC\_VER (string)

The version number of ODBC that CLI supports. By Default CLI returns "03.51". You can call the SQLSetEnvAttr() function to change the ODBC driver version. If you set the SQL\_ATTR\_ODBC\_VERSION attribute to SQL\_OV\_ODBC3\_80 (value 380), CLI returns "03.80".

### SQL\_DRIVER\_VER (string)

The version of the IBM Data Server Driver for ODBC and CLI. A string of the form 'mm.vv.rrrr' where *mm* is the major version, *vv* is the minor version, and *rrrr* is the release. For example, "05.01.0000" translates to major version 5, minor version 1, release 0. For full version information, use SQL\_DRIVER\_VER with SQL\_DRIVER\_BLDLEVEL.

### SQL\_DROP\_ASSERTION (32-bit unsigned integer)

Indicates which clause in the DROP ASSERTION statement is supported by the DBMS. CLI always returns zero; the DROP ASSERTION statement is not supported.

ODBC also defines the SQL\_DA\_DROP\_ASSERTION value that is not returned by CLI.

## SQLGetInfo function (CLI) - Get general information

### **SQL\_DROP\_CHARACTER\_SET (32-bit unsigned integer)**

Indicates which clause in the DROP CHARACTER SET statement is supported by the DBMS. CLI always returns zero; the DROP CHARACTER SET statement is not supported.

ODBC also defines the SQL\_DCS\_DROP\_CHARACTER\_SET value that is not returned by CLI.

### **SQL\_DROP\_COLLATION (32-bit unsigned integer)**

Indicates which clause in the DROP COLLATION statement is supported by the DBMS. CLI always returns zero; the DROP COLLATION statement is not supported.

ODBC also defines the SQL\_DC\_DROP\_COLLATION value that is not returned by CLI.

### **SQL\_DROP\_DOMAIN (32-bit unsigned integer)**

Indicates which clauses in the DROP DOMAIN statement are supported by the DBMS. CLI always returns zero; the DROP DOMAIN statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_DD\_DROP\_DOMAIN
- SQL\_DD\_CASCADE
- SQL\_DD\_RESTRICT

### **SQL\_DROP\_MODULE (32-bit unsigned integer)**

Indicates which clauses in the DROP MODULE statement are supported by the DBMS. CLI always returns zero for DB2 for z/OS.

CLI returns the listed values:

- SQL\_DM\_DROP\_MODULE
- SQL\_DM\_RESTRICT

### **SQL\_DROP\_SCHEMA (32-bit unsigned integer)**

Indicates which clauses in the DROP SCHEMA statement are supported by the DBMS. CLI always returns zero; the DROP SCHEMA statement is not supported.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_DS\_CASCADE
- SQL\_DS\_RESTRICT

### **SQL\_DROP\_TABLE (32-bit unsigned integer)**

Indicates which clauses in the DROP TABLE statement are supported by the DBMS. Valid returned values are:

- SQL\_DT\_DROP\_TABLE
- SQL\_DT\_CASCADE
- SQL\_DT\_RESTRICT

### **SQL\_DROP\_TRANSLATION (32-bit unsigned integer)**

Indicates which clauses in the DROP TRANSLATION statement are supported by the DBMS. CLI always returns zero; the DROP TRANSLATION statement is not supported.

ODBC also defines the listed value that is not returned by CLI:

- SQL\_DTR\_DROP\_TRANSLATION

### **SQL\_DROP\_VIEW (32-bit unsigned integer)**

Indicates which clauses in the DROP VIEW statement are supported by the DBMS. CLI always returns zero; the DROP VIEW statement is not supported.

## SQLGetInfo function (CLI) - Get general information

ODBC also defines the listed values that are not returned by CLI:

- SQL\_DV\_CASCADE
- SQL\_DV\_RESTRICT

### SQL\_DTC\_TRANSITION\_COST (32-bit unsigned mask)

Used by Microsoft Transaction Server to determine whether the enlistment process for a connection is expensive. CLI returns:

- SQL\_DTC\_ENLIST\_EXPENSIVE
- SQL\_DTC\_UNENLIST\_EXPENSIVE

### SQL\_DYNAMIC\_CURSOR\_ATTRIBUTES1 (32-bit mask)

Indicates the attributes of a dynamic cursor that are supported by CLI (subset 1 of 2). Valid returned values are:

- SQL\_CA1\_NEXT
- SQL\_CA1\_ABSOLUTE
- SQL\_CA1\_RELATIVE
- SQL\_CA1\_BOOKMARK
- SQL\_CA1\_LOCK\_EXCLUSIVE
- SQL\_CA1\_LOCK\_NO\_CHANGE
- SQL\_CA1\_LOCK\_UNLOCK
- SQL\_CA1\_POS\_POSITION
- SQL\_CA1\_POS\_UPDATE
- SQL\_CA1\_POS\_DELETE
- SQL\_CA1\_POS\_REFRESH
- SQL\_CA1\_POSITIONED\_UPDATE
- SQL\_CA1\_POSITIONED\_DELETE
- SQL\_CA1\_SELECT\_FOR\_UPDATE
- SQL\_CA1\_BULK\_ADD
- SQL\_CA1\_BULK\_UPDATE\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_DELETE\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_FETCH\_BY\_BOOKMARK

### SQL\_DYNAMIC\_CURSOR\_ATTRIBUTES2 (32-bit mask)

Indicates the attributes of a dynamic cursor that are supported by CLI (subset 2 of 2). Valid returned values are:

- SQL\_CA2\_READ\_ONLY\_CONCURRENCY
- SQL\_CA2\_LOCK\_CONCURRENCY
- SQL\_CA2\_OPT\_ROWVER\_CONCURRENCY
- SQL\_CA2\_OPT\_VALUES\_CONCURRENCY
- SQL\_CA2\_SENSITIVITY\_ADDITIONS
- SQL\_CA2\_SENSITIVITY\_DELETIONS
- SQL\_CA2\_SENSITIVITY\_UPDATES
- SQL\_CA2\_MAX\_ROWS\_SELECT
- SQL\_CA2\_MAX\_ROWS\_INSERT
- SQL\_CA2\_MAX\_ROWS\_DELETE
- SQL\_CA2\_MAX\_ROWS\_UPDATE
- SQL\_CA2\_MAX\_ROWS\_CATALOG
- SQL\_CA2\_MAX\_ROWS\_AFFECTS\_ALL
- SQL\_CA2\_CRC\_EXACT
- SQL\_CA2\_CRC\_APPROXIMATE
- SQL\_CA2\_SIMULATE\_NON\_UNIQUE
- SQL\_CA2\_SIMULATE\_TRY\_UNIQUE
- SQL\_CA2\_SIMULATE\_UNIQUE

### SQL\_EXPRESSIONS\_IN\_ORDERBY (string)

The character string "Y" indicates that the database server supports the DIRECT specification of expressions in the ORDER BY list, "N" indicates that it does not.

## SQLGetInfo function (CLI) - Get general information

### SQL\_FETCH\_DIRECTION (32-bit mask)

The supported fetch directions.

The listed bit masks are used with the flag to determine which options are supported:

- SQL\_FD\_FETCH\_NEXT
- SQL\_FD\_FETCH\_FIRST
- SQL\_FD\_FETCH\_LAST
- SQL\_FD\_FETCH\_PREV
- SQL\_FD\_FETCH\_ABSOLUTE
- SQL\_FD\_FETCH\_RELATIVE
- SQL\_FD\_FETCH\_RESUME

### SQL\_FILE\_USAGE (16-bit integer)

Indicates how a single-tier driver directly treats files in a data source. The IBM Data Server Driver for ODBC and CLI driver is not a single-tier driver, and therefore always returns SQL\_FILE\_NOT\_SUPPORTED.

ODBC also defines the listed values that are not returned by CLI:

- SQL\_FILE\_TABLE
- SQL\_FILE\_CATALOG

### SQL\_FORWARD\_ONLY\_CURSOR\_ATTRIBUTES1 (32-bit mask)

Indicates the attributes of a forward-only cursor that are supported by CLI. Valid returned values are (subset 1 of 2):

- SQL\_CA1\_NEXT
- SQL\_CA1\_POSITIONED\_UPDATE
- SQL\_CA1\_POSITIONED\_DELETE
- SQL\_CA1\_SELECT\_FOR\_UPDATE
- SQL\_CA1\_LOCK\_EXCLUSIVE
- SQL\_CA1\_LOCK\_NO\_CHANGE
- SQL\_CA1\_LOCK\_UNLOCK
- SQL\_CA1\_POS\_POSITION
- SQL\_CA1\_POS\_UPDATE
- SQL\_CA1\_POS\_DELETE
- SQL\_CA1\_POS\_REFRESH
- SQL\_CA1\_BULK\_ADD
- SQL\_CA1\_BULK\_UPDATE\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_DELETE\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_FETCH\_BY\_BOOKMARK

### SQL\_FORWARD\_ONLY\_CURSOR\_ATTRIBUTES2 (32-bit mask)

Indicates the attributes of a forward-only cursor that are supported by CLI. Valid returned values are (subset 2 of 2):

- SQL\_CA2\_READ\_ONLY\_CONCURRENCY
- SQL\_CA2\_LOCK\_CONCURRENCY
- SQL\_CA2\_MAX\_ROWS\_SELECT
- SQL\_CA2\_MAX\_ROWS\_CATALOG
- SQL\_CA2\_OPT\_ROWVER\_CONCURRENCY
- SQL\_CA2\_OPT\_VALUES\_CONCURRENCY
- SQL\_CA2\_SENSITIVITY\_ADDITIONS
- SQL\_CA2\_SENSITIVITY\_DELETIONS
- SQL\_CA2\_SENSITIVITY\_UPDATES
- SQL\_CA2\_MAX\_ROWS\_INSERT
- SQL\_CA2\_MAX\_ROWS\_DELETE
- SQL\_CA2\_MAX\_ROWS\_UPDATE
- SQL\_CA2\_MAX\_ROWS\_AFFECTS\_ALL
- SQL\_CA2\_CRC\_EXACT

## SQLGetInfo function (CLI) - Get general information

- SQL\_CA2\_CRC\_APPROXIMATE
- SQL\_CA2\_SIMULATE\_NON\_UNIQUE
- SQL\_CA2\_SIMULATE\_TRY\_UNIQUE
- SQL\_CA2\_SIMULATE\_UNIQUE

### SQL\_GETDATA\_EXTENSIONS (32-bit mask)

Indicates whether extensions to the SQLGetData() function are supported. The listed extensions are currently identified and supported by CLI:

- SQL\_GD\_ANY\_COLUMN, SQLGetData() can be called for unbound columns that precede the last bound column.
- SQL\_GD\_ANY\_ORDER, SQLGetData() can be called for columns in any order.

ODBC also defines the listed extensions which are not returned by CLI:

- SQL\_GD\_BLOCK
- SQL\_GD\_BOUND

### SQL\_GROUP\_BY (16-bit integer)

Indicates the degree of support for the GROUP BY clause by the server.

Valid returned values are:

- SQL\_GB\_NO\_RELATION - No relationship between the columns in the GROUP BY clause and in the SELECT list.
- SQL\_GB\_NOT\_SUPPORTED - GROUP BY clause not supported.
- SQL\_GB\_GROUP\_BY\_EQUALS\_SELECT - GROUP BY clause must include all non-aggregated columns in the SELECT list.
- SQL\_GB\_GROUP\_BY\_CONTAINS\_SELECT - GROUP BY clause must contain all non-aggregated columns in the SELECT list.
- SQL\_GB\_COLLATE - COLLATE clause can be specified at the end of each grouping column.

### SQL\_IDENTIFIER\_CASE (16-bit integer)

Indicates the case sensitivity of object names (such as table-name).

Valid returned values are::

- SQL\_IC\_UPPER : Stored in uppercase.
- SQL\_IC\_LOWER : Stored in lowercase.
- SQL\_IC\_SENSITIVE : Case sensitive, stored in mixed-case.
- SQL\_IC\_MIXED : Not case sensitive, stored in mixed-case.

**Note:** Identifier names in IBM DBMSs are not case sensitive.

### SQL\_IDENTIFIER\_QUOTE\_CHAR (string)

Indicates the character that is used to surround a delimited identifier.

### SQL\_INDEX\_KEYWORDS (32-bit mask)

Indicates the supported keywords for the CREATE INDEX statement. Valid returned values are:

- SQL\_IK\_NONE - None of the keywords are supported.
- SQL\_IK\_ASC - ASC keyword is supported.
- SQL\_IK\_DESC - DESC keyword is supported.
- SQL\_IK\_ALL - All keywords are supported.

To see if the CREATE INDEX statement is supported, an application can call the SQLGetInfo() function with the SQL\_DLL\_INDEX *InfoType* argument.

### SQL\_INFO\_SCHEMA\_VIEWS (32-bit mask)

Indicates the views in the INFORMATION\_SCHEMA that are supported. CLI always returns zero; no views in the INFORMATION\_SCHEMA are supported.

## SQLGetInfo function (CLI) - Get general information

ODBC also defines the listed values that are not returned by CLI:

- SQL\_ISV\_ASSERTIONS
- SQL\_ISV\_CHARACTER\_SETS
- SQL\_ISV\_CHECK\_CONSTRAINTS
- SQL\_ISV\_COLLATIONS
- SQL\_ISV\_COLUMN\_DOMAIN\_USAGE
- SQL\_ISV\_COLUMN\_PRIVILEGES
- SQL\_ISV\_COLUMNS
- SQL\_ISV\_CONSTRAINT\_COLUMN\_USAGE
- SQL\_ISV\_CONSTRAINT\_TABLE\_USAGE
- SQL\_ISV\_DOMAIN\_CONSTRAINTS
- SQL\_ISV\_DOMAINS
- SQL\_ISV\_KEY\_COLUMN\_USAGE
- SQL\_ISV\_REFERENTIAL\_CONSTRAINTS
- SQL\_ISV\_SCHEMATA
- SQL\_ISV\_SQL\_LANGUAGES
- SQL\_ISV\_TABLE\_CONSTRAINTS
- SQL\_ISV\_TABLE\_PRIVILEGES
- SQL\_ISV\_TABLES
- SQL\_ISV\_TRANSLATIONS
- SQL\_ISV\_USAGE\_PRIVILEGES
- SQL\_ISV\_VIEW\_COLUMN\_USAGE
- SQL\_ISV\_VIEW\_TABLE\_USAGE
- SQL\_ISV\_VIEWS

### SQL\_INSERT\_STATEMENT (32-bit mask)

Indicates support for INSERT statements. Valid returned values are:

- SQL\_IS\_INSERT\_LITERALS
- SQL\_IS\_INSERT\_SEARCHED
- SQL\_IS\_SELECT\_INTO

### SQL\_INTEGRITY (string)

The "Y" character string indicates that the data source supports Integrity Enhanced Facility (IEF) in SQL89 and in X/Open XPG4 Embedded SQL, an "N" indicates it does not.

In previous versions of CLI this *InfoType* argument was SQL\_ODBC\_SQL\_OPT\_IEF.

### SQL\_KEYSET\_CURSOR\_ATTRIBUTES1 (32-bit mask)

Indicates the attributes of a keyset-driven cursor that are supported by CLI. Valid returned values are (subset 1 of 2):

- SQL\_CA1\_NEXT
- SQL\_CA1\_ABSOLUTE
- SQL\_CA1\_RELATIVE
- SQL\_CA1\_BOOKMARK
- SQL\_CA1\_LOCK\_EXCLUSIVE
- SQL\_CA1\_LOCK\_NO\_CHANGE
- SQL\_CA1\_LOCK\_UNLOCK
- SQL\_CA1\_POS\_POSITION
- SQL\_CA1\_POS\_UPDATE
- SQL\_CA1\_POS\_DELETE
- SQL\_CA1\_POS\_REFRESH
- SQL\_CA1\_POSITIONED\_UPDATE
- SQL\_CA1\_POSITIONED\_DELETE
- SQL\_CA1\_SELECT\_FOR\_UPDATE
- SQL\_CA1\_BULK\_ADD
- SQL\_CA1\_BULK\_UPDATE\_BY\_BOOKMARK



## SQLGetInfo function (CLI) - Get general information

- SQL\_CA1\_BULK\_DELETE\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_FETCH\_BY\_BOOKMARK

### SQL\_KEYSET\_CURSOR\_ATTRIBUTES2 (32-bit mask)

Indicates the attributes of a keyset-driven cursor that are supported by CLI. Valid returned values are (subset 2 of 2):

- SQL\_CA2\_READ\_ONLY\_CONCURRENCY
- SQL\_CA2\_LOCK\_CONCURRENCY
- SQL\_CA2\_OPT\_ROWVER\_CONCURRENCY
- SQL\_CA2\_OPT\_VALUES\_CONCURRENCY
- SQL\_CA2\_SENSITIVITY\_ADDITIONS
- SQL\_CA2\_SENSITIVITY\_DELETIONS
- SQL\_CA2\_SENSITIVITY\_UPDATES
- SQL\_CA2\_MAX\_ROWS\_SELECT
- SQL\_CA2\_MAX\_ROWS\_INSERT
- SQL\_CA2\_MAX\_ROWS\_DELETE
- SQL\_CA2\_MAX\_ROWS\_UPDATE
- SQL\_CA2\_MAX\_ROWS\_CATALOG
- SQL\_CA2\_MAX\_ROWS\_AFFECTS\_ALL
- SQL\_CA2\_CRC\_EXACT
- SQL\_CA2\_CRC\_APPROXIMATE
- SQL\_CA2\_SIMULATE\_NON\_UNIQUE
- SQL\_CA2\_SIMULATE\_TRY\_UNIQUE
- SQL\_CA2\_SIMULATE\_UNIQUE

### SQL\_KEYWORDS (string)

Indicates a comma-separated list of all data source-specific keywords. This character string is a list of all reserved keywords. Interoperable applications should not use these keywords in object names. This list does not contain keywords specific to ODBC or keywords that are used by both the data source and ODBC.

### SQL\_LIKE\_ESCAPE\_CLAUSE (string)

Indicates whether the data source supports an escape character for the percent character (%) and underscore (\_) character in a LIKE predicate. Also, it indicates that the driver supports the ODBC syntax for defining a LIKE predicate escape character.

- "Y" indicates that there is support for escape characters in a LIKE predicate.
- "N" indicates that there is no support for escape characters in a LIKE predicate.

### SQL\_LOCK\_TYPES (32-bit mask)

Reserved option, zero is returned for the bit-mask.

### SQL\_MAX\_ASYNC\_CONCURRENT\_STATEMENTS (32-bit unsigned integer)

The maximum number of active concurrent statements in asynchronous mode that CLI can support on a given connection. This value is zero if there is no specific limit, or the limit is unknown.

### SQL\_MAX\_BINARY\_LITERAL\_LEN (32-bit unsigned integer)

A 32-bit unsigned integer value specifying the maximum length (number of hexadecimal characters, excluding the literal prefix and suffix returned by `SQLGetTypeInfo()`) of a binary literal in an SQL statement. For example, the binary literal `0xFFAA` has a length of 4. If there is no maximum length or the length is unknown, this value is set to zero.

## SQLGetInfo function (CLI) - Get general information

### SQL\_MAX\_CATALOG\_NAME\_LEN (16-bit integer)

The maximum length of a catalog name in the data source. This value is zero if there is no maximum length, or the length is unknown.

In previous versions of CLI this *InfoType* argument was SQL\_MAX\_QUALIFIER\_NAME\_LEN.

### SQL\_MAX\_CHAR\_LITERAL\_LEN (32-bit unsigned integer)

The maximum length of a character literal in an SQL statement (in bytes). Zero if there is no limit.

### SQL\_MAX\_COLUMN\_NAME\_LEN (16-bit integer)

The maximum length of a column name (in bytes). Zero if there is no limit.

### SQL\_MAX\_COLUMNS\_IN\_GROUP\_BY (16-bit integer)

Indicates the maximum number of columns that the server supports in a GROUP BY clause. Zero if there is no limit.

### SQL\_MAX\_COLUMNS\_IN\_INDEX (16-bit integer)

Indicates the maximum number of columns that the server supports in an index. Zero if there is no limit.

### SQL\_MAX\_COLUMNS\_IN\_ORDER\_BY (16-bit integer)

Indicates the maximum number of columns that the server supports in an ORDER BY clause. Zero if there is no limit.

### SQL\_MAX\_COLUMNS\_IN\_SELECT (16-bit integer)

Indicates the maximum number of columns that the server supports in a SELECT list. Zero if there no limit.

### SQL\_MAX\_COLUMNS\_IN\_TABLE (16-bit integer)

Indicates the maximum number of columns that the server supports in a base table. Zero if there is no limit.

### SQL\_MAX\_CONCURRENT\_ACTIVITIES (16-bit integer)

The maximum number of active environments that CLI can support. If there is no specified limit or the limit is unknown, this value is set to zero.

In previous versions of CLI this *InfoType* argument SQL\_ACTIVE\_ENVIRONMENTS.

### SQL\_MAX\_CURSOR\_NAME\_LEN (16-bit integer)

The maximum length of a cursor name (in bytes). This value is zero if there is no maximum length, or the length is unknown.

### SQL\_MAX\_DRIVER\_CONNECTIONS (16-bit integer)

The maximum number of active connections that are supported per application.

If the limit is dependent on system resources, zero is returned.

In previous versions of CLI this *InfoType* argument was SQL\_ACTIVE\_CONNECTIONS.

### SQL\_MAX\_IDENTIFIER\_LEN (16-bit integer)

The maximum size (in characters) that the data source supports for user-defined names.

### SQL\_MAX\_INDEX\_SIZE (32-bit unsigned integer)

Indicates the maximum size in bytes that the server supports for the combined columns in an index. Zero if no limit.

### SQL\_MAX\_MODULE\_NAME\_LEN (16-bit integer)

Indicates the maximum length in bytes of a module qualifier name.

## SQLGetInfo function (CLI) - Get general information

### SQL\_MAX\_PROCEDURE\_NAME\_LEN (16-bit integer)

The maximum length of a procedure name (in bytes).

### SQL\_MAX\_ROW\_SIZE (32-bit unsigned integer)

Specifies the maximum length in bytes that the server supports in single row of a base table. Zero if there is no limit.

### SQL\_MAX\_ROW\_SIZE\_INCLUDES\_LONG (string)

Set to "Y" to indicate that the value that is returned by `SQL_MAX_ROW_SIZE` *InfoType* argument includes the length of product-specific *long string* data types. Otherwise, set to "N".

### SQL\_MAX\_SCHEMA\_NAME\_LEN (16-bit integer)

The maximum length of a schema qualifier name (in bytes).

In previous versions of CLI this *InfoType* argument was `SQL_MAX_OWNER_NAME_LEN`.

### SQL\_MAX\_STATEMENT\_LEN (32-bit unsigned integer)

Indicates the maximum length of an SQL statement string in bytes, including the number of white spaces in the statement.

### SQL\_MAX\_TABLE\_NAME\_LEN (16-bit integer)

The maximum length of a table name (in bytes).

### SQL\_MAX\_TABLES\_IN\_SELECT (16-bit integer)

Indicates the maximum number of table names in a FROM clause in a <query specification>.

### SQL\_MAX\_USER\_NAME\_LEN (16-bit integer)

Indicates the maximum size for a <user identifier> (in bytes).

### SQL\_MODULE\_USAGE (32-bit mask)

Indicates the type of SQL statements that have a module associated with them when these statements are executed. CLI always returns zero for z/OS.

`SQL_MU_PROCEDURE_INVOCATION` is supported in the procedure invocation statement.

### SQL\_MULT\_RESULT\_SETS (string)

The character string "Y" indicates that the database supports multiple result sets, "N" indicates that it does not.

### SQL\_MULTIPLE\_ACTIVE\_TXN (string)

Indicates whether active transactions on multiple connections are permitted.

- "Y" indicates that multiple connections can have active transactions.
- "N" indicates that only one connection at a time can have an active transaction. CLI returns "N" for coordinated distributed unit of work (CONNECT TYPE 2) connections, (since the transaction or Unit Of Work spans all connections), and returns "Y" for all other connections.

### SQL\_NEED\_LONG\_DATA\_LEN (string)

Indicates that a character string is reserved for the use of ODBC. "N" is always returned.

### SQL\_NON\_NULLABLE\_COLUMNS (16-bit integer)

Indicates whether non-nullable columns are supported. Valid returned values are:

- `SQL_NNC_NON_NULL` - Can be defined as NOT NULL.
- `SQL_NNC_NULL` - Cannot be defined as NOT NULL.

## SQLGetInfo function (CLI) - Get general information

### SQL\_NULL\_COLLATION (16-bit integer)

Indicates where NULLs are sorted in a result set. Valid returned values are:

- SQL\_NC\_HIGH - Null values sort high.
- SQL\_NC\_LOW - Null values sort low.

### SQL\_NUMERIC\_FUNCTIONS (32-bit mask)

Indicates that the ODBC scalar numeric functions are supported. These functions are intended to be used with the ODBC vendor escape sequence.

The listed bit-masks are used to determine which numeric functions are supported:

- SQL\_FN\_NUM\_ABS
- SQL\_FN\_NUM\_ACOS
- SQL\_FN\_NUM\_ASIN
- SQL\_FN\_NUM\_ATAN
- SQL\_FN\_NUM\_ATAN2
- SQL\_FN\_NUM\_CEILING
- SQL\_FN\_NUM\_COS
- SQL\_FN\_NUM\_COT
- SQL\_FN\_NUM\_DEGREES
- SQL\_FN\_NUM\_EXP
- SQL\_FN\_NUM\_FLOOR
- SQL\_FN\_NUM\_LOG
- SQL\_FN\_NUM\_LOG10
- SQL\_FN\_NUM\_MOD
- SQL\_FN\_NUM\_PI
- SQL\_FN\_NUM\_POWER
- SQL\_FN\_NUM\_RADIANS
- SQL\_FN\_NUM\_RAND
- SQL\_FN\_NUM\_ROUND
- SQL\_FN\_NUM\_SIGN
- SQL\_FN\_NUM\_SIN
- SQL\_FN\_NUM\_SQRT
- SQL\_FN\_NUM\_TAN
- SQL\_FN\_NUM\_TRUNCATE

### SQL\_ODBC\_API\_CONFORMANCE (16-bit integer)

Indicates the level of ODBC conformance. Valid returned values are:

- SQL\_OAC\_NONE
- SQL\_OAC\_LEVEL1
- SQL\_OAC\_LEVEL2

### SQL\_ODBC\_INTERFACE\_CONFORMANCE (32-bit unsigned integer)

Indicates the level of the ODBC 3.0 interface that CLI conforms to:

- SQL\_OIC\_CORE : The minimum level that all ODBC drivers are expected to conform to. This level includes basic interface elements such as connection functions; functions for preparing and executing an SQL statement; basic result set metadata functions; basic catalog functions; and so on.
- SQL\_OIC\_LEVEL1 : A level that includes the core standards compliance level functionality, plus scrollable cursors, bookmarks, positioned updates and deletes; and so on.
- SQL\_OIC\_LEVEL2 : A level that includes the level 1 standards compliance level functionality, plus advanced features such as sensitive cursors; update, delete, and refresh by bookmarks; stored procedure support; catalog functions for primary and foreign keys; multi-catalog support; and so on.

## SQLGetInfo function (CLI) - Get general information

### SQL\_ODBC\_SAG\_CLI\_CONFORMANCE (16-bit integer)

The compliance to the functions of the SQL Access Group (SAG) CLI specification.

Valid returned values are:

- SQL\_OSCC\_NOT\_COMPLIANT : The driver is not SAG-compliant.
- SQL\_OSCC\_COMPLIANT : The driver is SAG-compliant.

### SQL\_ODBC\_SQL\_CONFORMANCE (16-bit integer)

Valid returned values are:

- SQL\_OSC\_MINIMUM : Minimum ODBC SQL grammar supported
- SQL\_OSC\_CORE : Core ODBC SQL Grammar supported
- SQL\_OSC\_EXTENDED : Extended ODBC SQL Grammar supported

### SQL\_ODBC\_VER (string)

The ODBC version number that the driver manager supports.

CLI returns the string "03.01.0000". CLI returns the string "03.01.0000". For Windows 7 and Windows Server 2008 R2 operating systems, CLI returns the string "03.80.0000".

### SQL\_OJ\_CAPABILITIES (32-bit mask)

A 32-bit bit-mask enumerating the types of outer join supported.

The bitmasks are:

- SQL\_OJ\_LEFT : Left outer join is supported.
- SQL\_OJ\_RIGHT : Right outer join is supported.
- SQL\_OJ\_FULL : Full outer join is supported.
- SQL\_OJ\_NESTED : Nested outer join is supported.
- SQL\_OJ\_ORDERED : The order of the tables underlying the columns in the outer join ON clause do not have to be in the same order as the tables in the JOIN clause.
- SQL\_OJ\_INNER : The inner table of an outer join can also be an inner join.
- SQL\_OJ\_ALL\_COMPARISONS\_OPS : Any predicate can be used in the outer join ON clause. If this bit is not set, only the equality (=) comparison operator can be used in outer joins.

### SQL\_ORDER\_BY\_COLUMNS\_IN\_SELECT (string)

Set to "Y" if columns in the ORDER BY clauses must be in the select list; otherwise set to "N".

### SQL\_OUTER\_JOINS (string)

The character string:

- "Y" indicates that outer joins are supported, and CLI supports the ODBC outer join request syntax.
- "N" indicates that outer joins are not supported.

### SQL\_PARAM\_ARRAY\_ROW\_COUNTS (32-bit unsigned integer)

Indicates the availability of row counts in a parameterized execution:

- SQL\_PARC\_BATCH : Individual row counts are available for each set of parameters. This behavior is conceptually equivalent to CLI generating a batch of SQL statements, one for each parameter set in the array.

Extended error information can be retrieved by using the SQL\_PARAM\_STATUS\_PTR descriptor field. To enable this behavior for non-atomic operations, set the SQL\_ATTR\_PARC\_BATCH connection attribute to SQL\_ATTR\_PARC\_BATCH\_ENABLE and SQL\_ATTR\_PARAMOPT\_ATOMIC to SQL\_ATTR\_PARAMOPT\_ATOMIC\_NO. If SQL\_ATTR\_PARAMOPT\_ATOMIC is set to SQL\_ATTR\_PARAMOPT\_ATOMIC\_YES, the CLI0150E error message is returned.

## SQLGetInfo function (CLI) - Get general information

- **SQL\_PARC\_NO\_BATCH** : Only one row count is available, which is the cumulative row count resulting from the execution of the statement for the entire array of parameters. This behavior is conceptually equivalent to treating the statement along with the entire parameter array as one atomic unit. Errors are handled the same as if one statement was issued.

### **SQL\_PARAM\_ARRAY\_SELECTS (32-bit unsigned integer)**

Indicates the availability of result sets in a parameterized execution. Valid returned values are:

- **SQL\_PAS\_BATCH** : One result set is available per set of parameters. The **SQL\_PAS\_BATCH** is conceptually equivalent to CLI generating a batch of SQL statements, one for each parameter set in the array.
- **SQL\_PAS\_NO\_BATCH** : Only one result set is available, which represents the cumulative result set resulting from the execution of the statement for the entire array of parameters. The **SQL\_PAS\_NO\_BATCH** is conceptually equivalent to treating the statement along with the entire parameter array as one atomic unit.
- **SQL\_PAS\_NO\_SELECT** : CLI does not allow a result-set generating statement to be executed with an array of parameters.

### **SQL\_POS\_OPERATIONS (32-bit mask)**

Reserved option, zero is returned for the bit-mask.

### **SQL\_POSITIONED\_STATEMENTS (32-bit mask)**

Indicates the degree of support for positioned UPDATE and positioned DELETE statements:

- **SQL\_PS\_POSITIONED\_DELETE**
- **SQL\_PS\_POSITIONED\_UPDATE**
- **SQL\_PS\_SELECT\_FOR\_UPDATE** - Indicates whether the server requires the FOR UPDATE clause to be specified on a <query expression> in order for a column to be updateable by using a cursor.

### **SQL\_PROCEDURE\_TERM (string)**

The name a database vendor uses for a procedure

### **SQL\_PROCEDURES (string)**

A character string of "Y" indicates that the data source supports procedures and CLI supports the ODBC procedure invocation syntax specified by the CALL statement. "N" indicates that it does not.

### **SQL\_QUOTED\_IDENTIFIER\_CASE (16-bit integer)**

Valid returned values are:

- **SQL\_IC\_UPPER** : Not case sensitive and are stored in uppercase.
- **SQL\_IC\_LOWER** : Not case sensitive and are stored in lowercase.
- **SQL\_IC\_SENSITIVE** : Quoted identifiers (delimited identifiers) in SQL are case sensitive and are stored in mixed case in the system catalog.
- **SQL\_IC\_MIXED** - Not case sensitive and are stored in mixed case.

The **SQL\_QUOTED\_IDENTIFIER\_CASE** integer should be contrasted with the **SQL\_IDENTIFIER\_CASE** *InfoType* argument, which is used to determine how (unquoted) identifiers are stored in the system catalog.

### **SQL\_ROW\_UPDATES (string)**

A character string of "Y" indicates a keyset-driven cursor or mixed cursor that maintains row versions or values for all fetched rows, and therefore can detect any updates made to a row since the row was last fetched. This character string only applies to updates, not to deletions or insertions. CLI can return the **SQL\_ROW\_UPDATED** flag to the row status array when **SQLFetchScroll()** is called. Otherwise, "N" is returned.

## SQLGetInfo function (CLI) - Get general information

### SQL\_SCHEMA\_TERM (string)

The terminology of the database vendor for a schema (owner).

In previous versions of CLI this *InfoType* was SQL\_OWNER\_TERM.

### SQL\_SCHEMA\_USAGE (32-bit mask)

Indicates the type of SQL statements that have schema (owners) associated with them when these statements are executed. Valid returned schema qualifiers (owners) are:

- SQL\_SU\_DML\_STATEMENTS - All DML statements.
- SQL\_SU\_PROCEDURE\_INVOCATION - The procedure invocation statement.
- SQL\_SU\_TABLE\_DEFINITION - All table definition statements.
- SQL\_SU\_INDEX\_DEFINITION - All index definition statements.
- SQL\_SU\_PRIVILEGE\_DEFINITION - All privilege definition statements (grant and revoke statements).

In previous versions of CLI this *InfoType* argument was SQL\_OWNER\_USAGE.

### SQL\_SCROLL\_CONCURRENCY (32-bit mask)

Indicates the concurrency options that are supported for the cursor.

The listed bit masks are used with the flag to determine which options are supported:

- SQL\_SCCO\_LOCK
- SQL\_SCCO\_READ\_ONLY
- SQL\_SCCO\_TIMESTAMP
- SQL\_SCCO\_VALUES

CLI returns SQL\_SCCO\_LOCK, which indicates that the lowest level of locking that is sufficient to make an update.

### SQL\_SCROLL\_OPTIONS (32-bit mask)

Indicates the scroll options that are supported for scrollable cursors.

The listed bit masks are used with the flag to determine which options are supported:

- SQL\_SO\_FORWARD\_ONLY : The cursor scrolls only forward.
- SQL\_SO\_KEYSET\_DRIVEN : CLI saves and uses the keys for every row in the result set.
- SQL\_SO\_STATIC : The data in the result set is static.
- SQL\_SO\_DYNAMIC : CLI keeps the keys for every row in the rowset (the keyset size is the same as the rowset size).
- SQL\_SO\_MIXED : CLI keeps the keys for every row in the keyset, and the keyset size is greater than the rowset size. The cursor is keyset-driven inside the keyset and dynamic outside the keyset.

### SQL\_SEARCH\_PATTERN\_ESCAPE (string)

Used to specify what the driver supports as an escape character for catalog functions, such as the SQLTables() function, and the SQLColumns() function.

### SQL\_SERVER\_NAME (string)

Indicates the name of the DB2 instance. In contrast to the SQL\_DATA\_SOURCE\_NAME character string, this character string is the actual name of the database server. Some DBMSs provide a different name upon establishing a connection than the real server-name of the database.

### SQL\_SPECIAL\_CHARACTERS (string)

A character string that contains only special characters (all characters

## SQLGetInfo function (CLI) - Get general information

except a...z, A...Z, 0...9, and underscore) that can be used in an identifier name, such as table, column, or index name, on the data source. For example, "@#". If an identifier contains special characters, the identifier must be a delimited identifier.

### SQL\_SQL\_CONFORMANCE (32-bit unsigned integer)

Indicates the level of SQL-92 that is supported:

- SQL\_SC\_SQL92\_ENTRY : Entry level SQL-92 compliant.
- SQL\_SC\_FIPS127\_2\_TRANSITIONAL : FIPS 127-2 transitional-level compliant.
- SQL\_SC\_SQL92\_FULL : Full-level SQL-92 compliant.
- SQL\_SC\_SQL92\_INTERMEDIATE : Intermediate level SQL-92 compliant.

### SQL\_SQL92\_DATETIME\_FUNCTIONS (32-bit mask)

Indicates the datetime scalar functions that are supported by CLI and the data source. Valid returned values are:

- SQL\_SDF\_CURRENT\_DATE
- SQL\_SDF\_CURRENT\_TIME
- SQL\_SDF\_CURRENT\_TIMESTAMP

### SQL\_SQL92\_FOREIGN\_KEY\_DELETE\_RULE (32-bit mask)

Indicates the rules that are supported for a foreign key in a DELETE statement, as defined by SQL-92. Valid returned values are:

- SQL\_SFKD\_CASCADE
- SQL\_SFKD\_NO\_ACTION
- SQL\_SFKD\_SET\_DEFAULT
- SQL\_SFKD\_SET\_NULL

### SQL\_SQL92\_FOREIGN\_KEY\_UPDATE\_RULE (32-bit mask)

Indicates the rules that are supported for a foreign key in an UPDATE statement, as defined by SQL-92. Valid returned values are:

- SQL\_SFKU\_CASCADE
- SQL\_SFKU\_NO\_ACTION
- SQL\_SFKU\_SET\_DEFAULT
- SQL\_SFKU\_SET\_NULL

### SQL\_SQL92\_GRANT (32-bit mask)

Indicates the clauses that are supported in a GRANT statement, as defined by SQL-92. Valid returned values are:

- SQL\_SG\_DELETE\_TABLE
- SQL\_SG\_INSERT\_COLUMN
- SQL\_SG\_INSERT\_TABLE
- SQL\_SG\_REFERENCES\_TABLE
- SQL\_SG\_REFERENCES\_COLUMN
- SQL\_SG\_SELECT\_TABLE
- SQL\_SG\_UPDATE\_COLUMN
- SQL\_SG\_UPDATE\_TABLE
- SQL\_SG\_USAGE\_ON\_DOMAIN
- SQL\_SG\_USAGE\_ON\_CHARACTER\_SET
- SQL\_SG\_USAGE\_ON\_COLLATION
- SQL\_SG\_USAGE\_ON\_TRANSLATION
- SQL\_SG\_WITH\_GRANT\_OPTION

### SQL\_SQL92\_NUMERIC\_VALUE\_FUNCTIONS (32-bit mask)

Indicates the numeric value scalar functions that are supported by CLI and the data source, as defined in SQL-92. Valid returned values are:

- SQL\_SNVF\_BIT\_LENGTH
- SQL\_SNVF\_CHAR\_LENGTH



## SQLGetInfo function (CLI) - Get general information

- SQL\_SNVF\_CHARACTER\_LENGTH
- SQL\_SNVF\_EXTRACT
- SQL\_SNVF\_OCTET\_LENGTH
- SQL\_SNVF\_POSITION

### SQL\_SQL92\_PREDICATES (32-bit mask)

Indicates the predicates that are supported in a SELECT statement, as defined by SQL-92. Valid returned values are:

- SQL\_SP\_BETWEEN
- SQL\_SP\_COMPARISON
- SQL\_SP\_EXISTS
- SQL\_SP\_IN
- SQL\_SP\_ISNOTNULL
- SQL\_SP\_ISNULL
- SQL\_SP\_LIKE
- SQL\_SP\_MATCH\_FULL
- SQL\_SP\_MATCH\_PARTIAL
- SQL\_SP\_MATCH\_UNIQUE\_FULL
- SQL\_SP\_MATCH\_UNIQUE\_PARTIAL
- SQL\_SP\_OVERLAPS
- SQL\_SP\_QUANTIFIED\_COMPARISON
- SQL\_SP\_UNIQUE

### SQL\_SQL92\_RELATIONAL\_JOIN\_OPERATORS (32-bit mask)

Indicates the relational join operators that are supported in a SELECT statement, as defined by SQL-92. Valid returned values are:

- SQL\_SRJO\_CORRESPONDING\_CLAUSE
- SQL\_SRJO\_CROSS\_JOIN
- SQL\_SRJO\_EXCEPT\_JOIN
- SQL\_SRJO\_FULL\_OUTER\_JOIN
- SQL\_SRJO\_INNER\_JOIN (indicates support for the INNER JOIN syntax, not for the inner join capability)
- SQL\_SRJO\_INTERSECT\_JOIN
- SQL\_SRJO\_LEFT\_OUTER\_JOIN
- SQL\_SRJO\_NATURAL\_JOIN
- SQL\_SRJO\_RIGHT\_OUTER\_JOIN
- SQL\_SRJO\_UNION\_JOIN

### SQL\_SQL92\_REVOKE (32-bit mask)

Indicates which clauses the data source supports in the REVOKE statement, as defined by SQL-92. Valid returned values are:

- SQL\_SR\_CASCADE
- SQL\_SR\_DELETE\_TABLE
- SQL\_SR\_GRANT\_OPTION\_FOR
- SQL\_SR\_INSERT\_COLUMN
- SQL\_SR\_INSERT\_TABLE
- SQL\_SR\_REFERENCES\_COLUMN
- SQL\_SR\_REFERENCES\_TABLE
- SQL\_SR\_RESTRICT
- SQL\_SR\_SELECT\_TABLE
- SQL\_SR\_UPDATE\_COLUMN
- SQL\_SR\_UPDATE\_TABLE
- SQL\_SR\_USAGE\_ON\_DOMAIN
- SQL\_SR\_USAGE\_ON\_CHARACTER\_SET
- SQL\_SR\_USAGE\_ON\_COLLATION
- SQL\_SR\_USAGE\_ON\_TRANSLATION

## SQLGetInfo function (CLI) - Get general information

### SQL\_SQL92\_ROW\_VALUE\_CONSTRUCTOR (32-bit mask)

Indicates the row value constructor expressions that are supported in a SELECT statement, as defined by SQL-92. Valid returned values are:

- SQL\_SRVC\_DEFAULT
- SQL\_SRVC\_NULL
- SQL\_SRVC\_ROW\_SUBQUERY
- SQL\_SRVC\_VALUE\_EXPRESSION

### SQL\_SQL92\_STRING\_FUNCTIONS (32-bit mask)

Indicates the string scalar functions that are supported by CLI and the data source, as defined by SQL-92. Valid returned values are:

- SQL\_SSF\_CONVERT
- SQL\_SSF\_LOWER
- SQL\_SSF\_SUBSTRING
- SQL\_SSF\_TRANSLATE
- SQL\_SSF\_TRIM\_BOTH
- SQL\_SSF\_TRIM\_LEADING
- SQL\_SSF\_TRIM\_TRAILING
- SQL\_SSF\_UPPER

### SQL\_SQL92\_VALUE\_EXPRESSIONS (32-bit mask)

Indicates the value expressions that are supported, as defined by SQL-92. Valid returned values are:

- SQL\_SVE\_CASE
- SQL\_SVE\_CAST
- SQL\_SVE\_COALESCE
- SQL\_SVE\_NULLIF

### SQL\_STANDARD\_CLI\_CONFORMANCE (32-bit mask)

Indicates the CLI standard or standards to which CLI conforms. Valid returned values are:

- SQL\_SCC\_ISO92\_CLI
- SQL\_SCC\_XOPEN\_CLI\_VERSION1

### SQL\_STATIC\_CURSOR\_ATTRIBUTES1 (32-bit mask)

Indicates the attributes of a static cursor that are supported by CLI. Valid returned values are (subset 1 of 2):

- SQL\_CA1\_ABSOLUTE
- SQL\_CA1\_BOOKMARK
- SQL\_CA1\_BULK\_ADD
- SQL\_CA1\_BULK\_DELETE\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_FETCH\_BY\_BOOKMARK
- SQL\_CA1\_BULK\_UPDATE\_BY\_BOOKMARK
- SQL\_CA1\_LOCK\_EXCLUSIVE
- SQL\_CA1\_LOCK\_NO\_CHANGE
- SQL\_CA1\_LOCK\_UNLOCK
- SQL\_CA1\_NEXT
- SQL\_CA1\_POS\_DELETE
- SQL\_CA1\_POS\_POSITION
- SQL\_CA1\_POS\_REFRESH
- SQL\_CA1\_POS\_UPDATE
- SQL\_CA1\_POSITIONED\_UPDATE
- SQL\_CA1\_POSITIONED\_DELETE
- SQL\_CA1\_RELATIVE
- SQL\_CA1\_SELECT\_FOR\_UPDATE

### SQL\_STATIC\_CURSOR\_ATTRIBUTES2 (32-bit mask)

Indicates the attributes of a static cursor that are supported by CLI. Valid returned values are (subset 2 of 2):

- SQL\_CA2\_CRC\_APPROXIMATE
- SQL\_CA2\_CRC\_EXACT
- SQL\_CA2\_LOCK\_CONCURRENCY
- SQL\_CA2\_OPT\_ROWVER\_CONCURRENCY
- SQL\_CA2\_OPT\_VALUES\_CONCURRENCY
- SQL\_CA2\_MAX\_ROWS\_AFFECTS\_ALL
- SQL\_CA2\_MAX\_ROWS\_CATALOG
- SQL\_CA2\_MAX\_ROWS\_DELETE
- SQL\_CA2\_MAX\_ROWS\_INSERT
- SQL\_CA2\_MAX\_ROWS\_SELECT
- SQL\_CA2\_MAX\_ROWS\_UPDATE
- SQL\_CA2\_READ\_ONLY\_CONCURRENCY
- SQL\_CA2\_SENSITIVITY\_ADDITIONS
- SQL\_CA2\_SENSITIVITY\_DELETIONS
- SQL\_CA2\_SENSITIVITY\_UPDATES
- SQL\_CA2\_SIMULATE\_NON\_UNIQUE
- SQL\_CA2\_SIMULATE\_TRY\_UNIQUE
- SQL\_CA2\_SIMULATE\_UNIQUE

### SQL\_STATIC\_SENSITIVITY (32-bit mask)

Indicates whether changes that are made by an application with a positioned update or delete statement can be detected by that application. Valid returned values are:

- SQL\_SS\_ADDITIONS : Added rows are visible to the cursor, and the cursor can scroll to these rows. All DB2 servers see added rows.
- SQL\_SS\_DELETIONS : Deleted rows are no longer available to the cursor, and do not leave a hole in the result set. After the cursor scrolls from a deleted row, it cannot return to that row.
- SQL\_SS\_UPDATES : Updated rows are visible to the cursor. If the cursor scrolls from and returns to an updated row, the data that is returned by the cursor is the updated data, not the original data.

### SQL\_STRING\_FUNCTIONS (32-bit mask)

Indicates which string functions are supported.

The listed bit masks are used to determine which string functions are supported:

- SQL\_FN\_STR\_ASCII
- SQL\_FN\_STR\_BIT\_LENGTH
- SQL\_FN\_STR\_CHAR
- SQL\_FN\_STR\_CHAR\_LENGTH
- SQL\_FN\_STR\_CHARACTER\_LENGTH
- SQL\_FN\_STR\_CONCAT
- SQL\_FN\_STR\_DIFFERENCE
- SQL\_FN\_STR\_INSERT
- SQL\_FN\_STR\_LCASE
- SQL\_FN\_STR\_LEFT
- SQL\_FN\_STR\_LENGTH
- SQL\_FN\_STR\_LOCATE
- SQL\_FN\_STR\_LOCATE\_2
- SQL\_FN\_STR\_LTRIM
- SQL\_FN\_STR\_OCTET\_LENGTH
- SQL\_FN\_STR\_POSITION
- SQL\_FN\_STR\_REPEAT

## SQLGetInfo function (CLI) - Get general information

- SQL\_FN\_STR\_REPLACE
- SQL\_FN\_STR\_RIGHT
- SQL\_FN\_STR\_RTRIM
- SQL\_FN\_STR\_SOUNDEX
- SQL\_FN\_STR\_SPACE
- SQL\_FN\_STR\_SUBSTRING
- SQL\_FN\_STR\_UCASE

If an application can call the LOCATE scalar function with the *string\_exp1*, *string\_exp2*, and *start* arguments, the SQL\_FN\_STR\_LOCATE bit mask is returned. If an application can call the LOCATE scalar function only with the *string\_exp1* and *string\_exp2*, the SQL\_FN\_STR\_LOCATE\_2 bit mask is returned. If the LOCATE scalar function is fully supported, both bit masks are returned.

### SQL\_SUBQUERIES (32-bit mask)

Indicates which predicates support subqueries. Valid returned values are:

- SQL\_SQ\_COMPARISON : The *comparison* predicate.
- SQL\_SQ\_CORRELATE\_SUBQUERIES : All predicates that support subqueries also support correlated subqueries.
- SQL\_SQ\_EXISTS : The *exists* predicate.
- SQL\_SQ\_IN : The *in* predicate.
- SQL\_SQ\_QUANTIFIED : The predicates that contains a quantification scalar function.

### SQL\_SYSTEM\_FUNCTIONS (32-bit mask)

Indicates which scalar system functions are supported.

The listed bit masks are used to determine which scalar system functions are supported:

- SQL\_FN\_SYS\_DBNAME
- SQL\_FN\_SYS\_IFNULL
- SQL\_FN\_SYS\_USERNAME

**Note:** These functions are intended to be used with the escape sequence in ODBC.

### SQL\_TABLE\_TERM (string)

The terminology of a database vendor for a table.

### SQL\_TIMEDATE\_ADD\_INTERVALS (32-bit mask)

Indicates whether or not the special ODBC system function TIMESTAMPADD is supported, and, if it is, which intervals are supported.

The listed bit masks are used to determine which intervals are supported:

- SQL\_FN\_TSI\_FRAC\_SECOND
- SQL\_FN\_TSI\_SECOND
- SQL\_FN\_TSI\_MINUTE
- SQL\_FN\_TSI\_HOUR
- SQL\_FN\_TSI\_DAY
- SQL\_FN\_TSI\_WEEK
- SQL\_FN\_TSI\_MONTH
- SQL\_FN\_TSI\_QUARTER
- SQL\_FN\_TSI\_YEAR

### SQL\_TIMEDATE\_DIFF\_INTERVALS (32-bit mask)

Indicates whether or not the special ODBC system function TIMESTAMPDIF is supported, and, if it is, which intervals are supported.

The listed bit masks are used to determine which intervals are supported:

## SQLGetInfo function (CLI) - Get general information

- SQL\_FN\_TSI\_FRAC\_SECOND
- SQL\_FN\_TSI\_SECOND
- SQL\_FN\_TSI\_MINUTE
- SQL\_FN\_TSI\_HOUR
- SQL\_FN\_TSI\_DAY
- SQL\_FN\_TSI\_WEEK
- SQL\_FN\_TSI\_MONTH
- SQL\_FN\_TSI\_QUARTER
- SQL\_FN\_TSI\_YEAR

### SQL\_TIMEDATE\_FUNCTIONS (32-bit mask)

Indicates which time and date functions are supported.

The listed bit masks are used to determine which date functions are supported:

- SQL\_FN\_TD\_CURRENT\_DATE
- SQL\_FN\_TD\_CURRENT\_TIME
- SQL\_FN\_TD\_CURRENT\_TIMESTAMP
- SQL\_FN\_TD\_CURDATE
- SQL\_FN\_TD\_CURTIME
- SQL\_FN\_TD\_DAYNAME
- SQL\_FN\_TD\_DAYOFMONTH
- SQL\_FN\_TD\_DAYOFWEEK
- SQL\_FN\_TD\_DAYOFYEAR
- SQL\_FN\_TD\_EXTRACT
- SQL\_FN\_TD\_HOUR
- SQL\_FN\_TD\_JULIAN\_DAY
- SQL\_FN\_TD\_MINUTE
- SQL\_FN\_TD\_MONTH
- SQL\_FN\_TD\_MONTHNAME
- SQL\_FN\_TD\_NOW
- SQL\_FN\_TD\_QUARTER
- SQL\_FN\_TD\_SECOND
- SQL\_FN\_TD\_SECONDS\_SINCE\_MIDNIGHT
- SQL\_FN\_TD\_TIMESTAMPADD
- SQL\_FN\_TD\_TIMESTAMPDIFF
- SQL\_FN\_TD\_WEEK
- SQL\_FN\_TD\_YEAR

**Note:** These functions are intended to be used with the escape sequence in ODBC.

### SQL\_TXN\_CAPABLE (16-bit integer)

Indicates whether transactions can contain DDL, DML, or both. Valid returned values are:

- SQL\_TC\_NONE : Transactions not supported.
- SQL\_TC\_DML : Transactions can contain only DML statements (for example, SELECT, INSERT, UPDATE and DELETE). DDL statements, such as CREATE TABLE and DROP INDEX, that are encountered in a transaction cause an error.
- SQL\_TC\_DDL\_COMMIT : Transactions can only contain DML statements. DDL statements that are encountered in a transaction cause the transaction to be committed.
- SQL\_TC\_DDL\_IGNORE : Transactions can only contain DML statements. DDL statements that are encountered in a transaction are ignored.
- SQL\_TC\_ALL : Transactions can contain DDL and DML statements in any order.

## SQLGetInfo function (CLI) - Get general information

### SQL\_TXN\_ISOLATION\_OPTION (32-bit mask)

The transaction isolation levels that are available at the currently connected database server.

The listed masks are used in conjunction with the flag to determine which options are supported:

- SQL\_TXN\_NOCOMMIT
- SQL\_TXN\_READ\_COMMITTED
- SQL\_TXN\_READ\_UNCOMMITTED
- SQL\_TXN\_REPEATABLE\_READ
- SQL\_TXN\_SERIALIZABLE
- SQL\_TXN\_VERSIONING

For descriptions of each level, see SQL\_DEFAULT\_TXN\_ISOLATION.

### SQL\_UNION (32-bit mask)

Indicates if the server supports the UNION operator. Valid returned values are:

- SQL\_U\_UNION : Supports the UNION clause.
- SQL\_U\_UNION\_ALL : Supports the ALL keyword in the UNION clause.

If SQL\_U\_UNION\_ALL is set, so is SQL\_U\_UNION.

### SQL\_USER\_NAME (string)

Indicates the user name that is used in a particular database. This character string is the identifier that is specified on the SQLConnect() call.

### SQL\_XOPEN\_CLI\_YEAR (string)

Indicates the year of publication of the X/Open specification with which the version of the driver fully complies.

---

## SQLGetLength function (CLI) - Retrieve length of a string value

### Purpose

Specification:	CLI 2.1		
----------------	---------	--	--

SQLGetLength() is used to retrieve the length of a large object value, referenced by a large object locator that has been returned from the server (as a result of a fetch, or an SQLGetSubString() call) during the current transaction.

### Syntax

```
SQLRETURN SQLGetLength (SQLHSTMT          StatementHandle, /* hstmt */
                        SQLSMALLINT       LocatorCType,
                        SQLINTEGER         Locator,
                        SQLINTEGER         *StringLength,
                        SQLINTEGER         *IndicatorValue);
```

### Function arguments

Table 95. SQLGetLength arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.

## SQLGetLength function (CLI) - Retrieve length of a string value

Table 95. *SQLGetLength* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>LocatorCType</i>	input	The C type of the source LOB locator. This may be: <ul style="list-style-type: none"> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> </ul>
SQLINTEGER	<i>Locator</i>	input	Must be set to the LOB locator value.
SQLINTEGER *	<i>StringLength</i>	output	The length of the returned information in <i>rgbValue</i> in bytes <sup>a</sup> if the target C buffer type is intended for a binary or character string variable and not a locator value.  If the pointer is set to NULL then the SQLSTATE HY009 is returned.
SQLINTEGER *	<i>IndicatorValue</i>	output	Always set to zero.

**Note:**

**a** This is in characters for DBCLOB data.

### Usage

SQLGetLength() can be used to determine the length of the data value represented by a LOB locator. It is used by applications to determine the overall length of the referenced LOB value so that the appropriate strategy to obtain some or all of the LOB value can be chosen. The length is calculated by the database server using the server code page, and so if the application code page is different from the server code page, then there may be some complexity in calculating space requirements on the client. The application will need to allow for code page expansion if any is needed.

The *Locator* argument can contain any valid LOB locator which has not been explicitly freed using a FREE LOCATOR statement nor implicitly freed because the transaction during which it was created has ended.

The statement handle must not have been associated with any prepared statements or catalog function calls.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 96. *SQLGetLength* SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The combination of <i>LocatorCType</i> and <i>Locator</i> is not valid.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.

## SQLGetLength function (CLI) - Retrieve length of a string value

Table 96. SQLGetLength SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	<i>LocatorCType</i> is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY009	Invalid argument value.	Pointer to <i>StringLength</i> was NULL.
HY010	Function sequence error.	The specified <i>StatementHandle</i> is not in an <i>allocated</i> state.  The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	The LOB token variable does not currently represent any value.	The value specified for <i>Locator</i> has not been associated with a LOB locator.

### Restrictions

This function is not available when connected to a DB2 server that does not support large objects. Call SQLGetFunctions() with the function type set to SQL\_API\_SQLGETLENGTH and check the *fExists* output argument to determine if the function is supported for the current connection.

### Example

```
/* get the length of the whole CLOB data */
cliRC = SQLGetLength(hstmtLocUse,
                    SQL_C_CLOB_LOCATOR,
                    clobLoc,
                    &clobLen,
                    &ind);
```

## SQLGetPosition function (CLI) - Return starting position of string

### Purpose

Specification:	CLI 2.1		
----------------	---------	--	--

SQLGetPosition() is used to return the starting position of one string within a LOB value (the source). The source value must be a LOB locator, the search string can be a LOB locator or a literal string.



## SQLGetPosition function (CLI) - Return starting position of string

The source and search LOB locators can be any that have been returned from the database from a fetch or a SQLGetSubString() call during the current transaction.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetPositionW(). For information about ANSI to Unicode function mappings, refer to “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLGetPosition (SQLHSTMT      StatementHandle, /* hstmt */
                          SQLSMALLINT   LocatorCType,
                          SQLINTEGER     SourceLocator,
                          SQLINTEGER     SearchLocator,
                          SQLCHAR        *SearchLiteral,
                          SQLINTEGER     SearchLiteralLength,
                          SQLUINTEGER    FromPosition,
                          SQLUINTEGER    *LocatedAt,
                          SQLINTEGER     *IndicatorValue);
```

### Function arguments

Table 97. SQLGetPosition arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	<i>LocatorCType</i>	input	The C type of the source LOB locator. This can be: <ul style="list-style-type: none"> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> </ul>
SQLINTEGER	<i>Locator</i>	input	<i>Locator</i> must be set to the source LOB locator.
SQLINTEGER	<i>SearchLocator</i>	input	If the <i>SearchLiteral</i> pointer is NULL and if <i>SearchLiteralLength</i> is set to 0, then <i>SearchLocator</i> must be set to the LOB locator associated with the search string; otherwise, this argument is ignored.
SQLCHAR *	<i>SearchLiteral</i>	input	This argument points to the area of storage that contains the search string literal.  If <i>SearchLiteralLength</i> is 0, this pointer must be NULL.
SQLINTEGER	<i>SearchLiteralLength</i>	input	The length of the string in <i>SearchLiteral</i> (in bytes). <sup>a</sup> The length of the string in <i>SearchLiteral</i> (in bytes) will be the number of WCHAR characters for the Unicode variant of this function.  If this argument value is 0, then the argument <i>SearchLocator</i> is meaningful.
SQLUINTEGER	<i>FromPosition</i>	input	For BLOBs and CLOBs, this is the position of the first byte within the source string at which the search is to start. For DBCLOBs, this is the first character. The start byte or character is numbered 1.
SQLUINTEGER *	<i>LocatedAt</i>	output	For BLOBs and CLOBs, this is the byte position at which the string was located or, if not located, the value zero. For DBCLOBs, this is the character position.  If the length of the source string is zero, the value 1 is returned.

## SQLGetPosition function (CLI) - Return starting position of string

Table 97. SQLGetPosition arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	<i>IndicatorValue</i>	output	Always set to zero.

### Note:

a This is in bytes even for DBCLOB data.

## Usage

SQLGetPosition() is used in conjunction with SQLGetSubString() in order to obtain any portion of a LOB in a random manner. In order to use SQLGetSubString(), the location of the substring within the overall string must be known in advance. In situations where the start of that substring can be found by a search string, SQLGetPosition() can be used to obtain the starting position of that substring.

The *Locator* and *SearchLocator* (if used) arguments can contain any valid LOB locator which has not been explicitly freed using a FREE LOCATOR statement or implicitly freed because the transaction during which it was created has ended.

The *Locator* and *SearchLocator* must have the same LOB locator type.

The statement handle must not have been associated with any prepared statements or catalog function calls.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 98. SQLGetPosition SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The combination of <i>LocatorCType</i> and either of the LOB locator values is not valid.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	The pointer to the <i>LocatedAt</i> argument was NULL.  The argument value for <i>FromPosition</i> was not greater than 0.  <i>LocatorCType</i> is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.

## SQLGetPosition function (CLI) - Return starting position of string

Table 98. SQLGetPosition SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The specified <i>StatementHandle</i> is not in an <i>allocated</i> state.</p> <p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p>
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	<p>The value of <i>SearchLiteralLength</i> was less than 1, and not SQL_NTS.</p> <p>The length of the pattern is longer than the maximum data length of the associated variable SQL data type (for DB2 for z/OS servers, the pattern length is a maximum of 4000 bytes regardless of the data type or the <i>LocatorCType</i>). For <i>LocatorCType</i> of SQL_C_CLOB_LOCATOR, the literal maximum size is that of an SQLCLOB; for <i>LocatorCType</i> of SQL_C_BLOB_LOCATOR, the literal maximum size is that of an SQLVARBINARY; for <i>LocatorCType</i> of SQL_C_DBCLOB_LOCATOR, the literal maximum size is that of an SQLVARGRAPHIC.</p>
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	The LOB token variable does not currently represent any value.	The value specified for <i>Locator</i> or <i>SearchLocator</i> is not currently a LOB locator.

### Restrictions

This function is not available when connected to a DB2 server that does not support large objects. Call SQLGetFunctions() with the function type set to SQL\_API\_SQLGETPOSITION and check the *fExists* output argument to determine if the function is supported for the current connection.

The SQLGetPosition() function is intended to handle graphic data and not WCHAR data. As a result, the data passed to this function should be considered big endian.

### Example

```
/* get the starting position of the CLOB piece of data */
cliRC = SQLGetPosition(hstmtLocUse,
                      SQL_C_CLOB_LOCATOR,
                      clobLoc,
                      0,
                      (SQLCHAR *)"Interests",
                      strlen("Interests"),
                      1,
                      &clobPiecePos,
                      &ind);
```

## SQLGetSQLCA function (CLI) - Get SQLCA data structure

### Deprecated

#### Note:

SQLGetSQLCA() has been deprecated.

Although this version of CLI continues to support SQLGetSQLCA(), it is recommended that you stop using it in your CLI programs so that they conform to the latest standards.

Use SQLGetDiagField() and SQLGetDiagRec() to retrieve diagnostic information.

## SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLGetStmtAttr() returns the current setting of a statement attribute.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetStmtAttrW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLGetStmtAttr (SQLHSTMT StatementHandle,
                          SQLINTEGER Attribute,
                          SQLPOINTER ValuePtr,
                          SQLINTEGER BufferLength,
                          SQLINTEGER *StringLengthPtr);
```

### Function arguments

Table 99. SQLGetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLINTEGER	<i>Attribute</i>	input	Attribute to retrieve.
SQLPOINTER	<i>ValuePtr</i>	output	Pointer to a buffer in which to return the value of the attribute specified in <i>Attribute</i> .

## SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute

Table 99. SQLGetStmtAttr arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>BufferLength</i>	input	<p>If <i>Attribute</i> is an ODBC-defined attribute and <i>ValuePtr</i> points to a character string or a binary buffer, this argument should be the length of <i>*ValuePtr</i>. If <i>Attribute</i> is an ODBC-defined attribute and <i>*ValuePtr</i> is an integer, <i>BufferLength</i> is ignored.</p> <p>If <i>Attribute</i> is a CLI attribute, the application indicates the nature of the attribute by setting the <i>BufferLength</i> argument. <i>BufferLength</i> can have the following values:</p> <ul style="list-style-type: none"> <li>• If <i>*ValuePtr</i> is a pointer to a character string, then <i>BufferLength</i> is the number of bytes needed to store the string, or SQL_NTS.</li> <li>• If <i>*ValuePtr</i> is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in <i>BufferLength</i>. This places a negative value in <i>BufferLength</i>.</li> <li>• If <i>*ValuePtr</i> is a pointer to a value other than a character string or binary string, then <i>BufferLength</i> should have the value SQL_IS_POINTER.</li> <li>• If <i>*ValuePtr</i> contains a fixed-length data type, then <i>BufferLength</i> is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.</li> <li>• If the value returned in <i>ValuePtr</i> is a Unicode string, the <i>BufferLength</i> argument must be an even number.</li> </ul>
SQLSMALLINT *	<i>StringLengthPtr</i>	output	<p>A pointer to a buffer in which to return the total number of bytes (excluding the null termination character) available to return in <i>*ValuePtr</i>. If this is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than or equal to <i>BufferLength</i>, the data in <i>*ValuePtr</i> is truncated to <i>BufferLength</i> minus the length of a null termination character and is null-terminated by the CLI.</p>

### Usage

A call to SQLGetStmtAttr() returns in *\*ValuePtr* the value of the statement attribute specified in *Attribute*. That value can either be a 32-bit value or a null-terminated character string. If the value is a null-terminated string, the application specifies the maximum length of that string in the *BufferLength* argument, and CLI returns the length of that string in the *\*StringLengthPtr* buffer. If the value is a 32-bit value, the *BufferLength* and *StringLengthPtr* arguments are not used.

The following statement attributes are read-only, so can be retrieved by SQLGetStmtAttr(), but not set by SQLSetStmtAttr(). Refer to the list of statement attributes for all statement attributes that can be set and retrieved.

- SQL\_ATTR\_IMP\_PARAM\_DESC
- SQL\_ATTR\_IMP\_ROW\_DESC
- SQL\_ATTR\_ROW\_NUMBER

## SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 100. SQLGetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The data returned in <i>*ValuePtr</i> was truncated to be <i>BufferLength</i> minus the length of a null termination character. The length of the untruncated string value is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state.	The argument <i>Attribute</i> was SQL_ATTR_ROW_NUMBER and the cursor was not open, or the cursor was positioned before the start of the result set or after the end of the result set.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called.  SQLExecute() or SQLExecDirect() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument <i>BufferLength</i> was less than 0.
HY092	Option type out of range.	The value specified for the argument <i>Attribute</i> was not valid for this version of CLI
HY109	Invalid cursor position.	The <i>Attribute</i> argument was SQL_ATTR_ROW_NUMBER and the row had been deleted or could not be fetched.
HYC00	Driver not capable.	The value specified for the argument <i>Attribute</i> was a valid CLI attribute for the version of CLI, but was not supported by the data source.

### Restrictions

None.

## SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute

### Example

```
/* get the handle for the implicitly allocated descriptor */
rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_IMP_ROW_DESC,
                    &hIRD,
                    SQL_IS_INTEGER,
                    &indicator);
```

---

## SQLGetStmtOption function (CLI) - Return current setting of a statement option

### Deprecated

#### Note:

In ODBC 3.0, SQLGetStmtOption() has been deprecated and replaced with SQLGetStmtAttr().

Although this version of CLI continues to support SQLGetStmtOption(), use SQLGetStmtAttr() in your CLI programs so that they conform to the latest standards.

### Migrating to the new function

The statement:

```
SQLGetStmtOption(hstmt, SQL_ATTR_CURSOR_HOLD, pvCursorHold);
```

for example, would be rewritten using the new function as:

```
SQLGetStmtAttr(hstmt, SQL_ATTR_CURSOR_HOLD, pvCursorHold,
               SQL_IS_INTEGER, NULL);
```

---

## SQLGetSubString function (CLI) - Retrieve portion of a string value

### Purpose

Specification:	CLI 2.1		
----------------	---------	--	--

SQLGetSubString() is used to retrieve a portion of a large object value, referenced by a large object locator that has been returned from the server (returned by a fetch or a previous SQLGetSubString() call) during the current transaction.

### Syntax

```
SQLRETURN SQLGetSubString (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLSMALLINT       LocatorCType,
    SQLINTEGER         SourceLocator,
    SQLUINTEGER        FromPosition,
    SQLUINTEGER        ForLength,
    SQLSMALLINT       TargetCType,
    SQLPOINTER         DataPtr,        /* rgbValue */
    SQLINTEGER         BufferLength,    /* cbValueMax */
    SQLINTEGER         *StringLength,
    SQLINTEGER         *IndicatorValue);
```

## SQLGetSubString function (CLI) - Retrieve portion of a string value

### Function arguments

Table 101. SQLGetSubString arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	<i>LocatorCType</i>	input	The C type of the source LOB locator. This may be: <ul style="list-style-type: none"> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> </ul>
SQLINTEGER	<i>Locator</i>	input	<i>Locator</i> must be set to the source LOB locator value.
SQLINTEGER	<i>FromPosition</i>	input	For BLOBs and CLOBs, this is the position of the first byte to be returned by the function. For DBCLOBs, this is the first character. The start byte or character is numbered 1.
SQLINTEGER	<i>ForLength</i>	input	This is the length of the string to be returned by the function. For BLOBs and CLOBs, this is the length in bytes. For DBCLOBs, this is the length in characters.  If <i>FromPosition</i> is less than the length of the source string but <i>FromPosition</i> + <i>ForLength</i> - 1 extends beyond the end of the source string, the result is padded on the right with the necessary number of characters (X'00' for BLOBs, single byte blank character for CLOBs, and double byte blank character for DBCLOBs).
SQLSMALLINT	<i>TargetCType</i>	input	The C data type of the <i>DataPtr</i> . The target must always be either a LOB locator C buffer type: <ul style="list-style-type: none"> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> </ul> or a C string type: <ul style="list-style-type: none"> <li>• SQL_C_CHAR</li> <li>• SQL_C_WCHAR</li> <li>• SQL_C_BINARY</li> <li>• SQL_C_DBCHAR</li> </ul>
SQLPOINTER	<i>DataPtr</i>	output	Pointer to the buffer where the retrieved string value or a LOB locator is to be stored.
SQLINTEGER	<i>BufferLength</i>	input	Maximum size of the buffer pointed to by <i>DataPtr</i> in bytes.
SQLINTEGER *	<i>StringLength</i>	output	The length of the returned information in <i>DataPtr</i> in bytes <sup>a</sup> if the target C buffer type is intended for a binary or character string variable and not a locator value.  If the pointer is set to NULL, nothing is returned.
SQLINTEGER *	<i>IndicatorValue</i>	output	Always set to zero.

**Note:**

**a** This is in bytes even for DBCLOB data.



## SQLGetSubString function (CLI) - Retrieve portion of a string value

### Usage

SQLGetSubString() is used to obtain any portion of the string that is represented by the LOB locator. There are two choices for the target:

- The target can be an appropriate C string variable.
- A new LOB value can be created on the server and the LOB locator for that value can be assigned to a target application variable on the client.

SQLGetSubString() can be used as an alternative to SQLGetData() for getting LOB data in pieces. In this case a column is first bound to a LOB locator, which is then used to fetch the LOB as a whole or in pieces.

The Locator argument can contain any valid LOB locator which has not been explicitly freed using a FREE LOCATOR statement nor implicitly freed because the transaction during which it was created has ended.

The statement handle must not have been associated with any prepared statements or catalog function calls.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 102. SQLGetSubString SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The amount of data to be returned is longer than <i>BufferLength</i> . The actual length of data available for return is stored in <i>StringLength</i> .
07006	Invalid conversion.	The value specified for <i>TargetCType</i> was not SQL_C_CHAR, SQL_WCHAR, SQL_C_BINARY, SQL_C_DBCHAR, or a LOB locator.  The value specified for <i>TargetCType</i> is inappropriate for the source (for example SQL_C_DBCHAR for a BLOB column).
22011	A substring error occurred.	<i>FromPosition</i> is greater than the of length of the source string.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	<i>LocatorCType</i> is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY009	Invalid argument value.	The value specified for <i>FromPosition</i> or for <i>ForLength</i> was not a positive integer.

## SQLGetSubString function (CLI) - Retrieve portion of a string value

Table 102. SQLGetSubString SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The specified <i>StatementHandle</i> is not in an <i>allocated</i> state.  The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value of <i>BufferLength</i> was less than 0.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	No locator currently assigned	The value specified for <i>Locator</i> is not currently a LOB locator.

### Restrictions

This function is not available when connected to a DB2 server that does not support large objects. Call SQLGetFunctions() with the function type set to SQL\_API\_SQLGETSUBSTRING and check the *fExists* output argument to determine if the function is supported for the current connection.

*FromPosition* of SQLGetSubString() can not take zero or negative values for IDS Data Servers. This is a current limitation.

When accessing IDS data servers, calling the SQLGetSubString() function with a TargetCType argument value of SQL\_C\_CLOB\_LOCATOR or SQL\_C\_BLOB\_LOCATOR will return an "Invalid Conversion" error. This conversion is not supported.

### Example

```
/* read the piece of CLOB data in buffer */
cliRC = SQLGetSubString(hstmtLocUse,
                        SQL_C_CLOB_LOCATOR,
                        clobLoc,
                        clobPiecePos,
                        clobLen - clobPiecePos,
                        SQL_C_CHAR,
                        buffer,
                        clobLen - clobPiecePos + 1,
                        &clobPieceLen,
                        &ind);
```

## SQLGetTypeInfo function (CLI) - Get data type information

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

## SQLGetTypeInfo function (CLI) - Get data type information

SQLGetTypeInfo() returns information about the data types that are supported by the DBMSs associated with CLI. The information is returned in an SQL result set. The columns can be received using the same functions that are used to process a query.

### Syntax

```
SQLRETURN SQLGetTypeInfo (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLSMALLINT      DataType);      /* fSqlType */
```

### Function arguments

Table 103. SQLGetTypeInfo arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLSMALLINT	<i>DataType</i>	input	<p>The SQL data type being queried. The supported types are:</p> <ul style="list-style-type: none"> <li>• SQL_ALL_TYPES</li> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BIT</li> <li>• SQL_BLOB</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_DATE <sup>1</sup></li> <li>• SQL_TYPE_DATE</li> <li>• SQL_DBCLOB</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_LONGVARBINARY</li> <li>• SQL_LONGVARCHAR</li> <li>• SQL_LONGVARGRAPHIC</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TIME <sup>1</sup></li> <li>• SQL_TIMESTAMP <sup>1</sup></li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_TINYINT</li> <li>• SQL_VARBINARY</li> <li>• SQL_VARCHAR</li> <li>• SQL_VARGRAPHIC</li> <li>• SQL_XML</li> </ul> <p>If SQL_ALL_TYPES is specified, information about all supported data types would be returned in ascending order by TYPE_NAME. All unsupported data types would be absent from the result set.</p>

### Note:

1. These SQL data types are supported for compatibility with ODBC 2.0.

## SQLGetTypeInfo function (CLI) - Get data type information

### Usage

Since `SQLGetTypeInfo()` generates a result set and is equivalent to executing a query, it will generate a cursor and begin a transaction. To prepare and execute another statement on this statement handle, the cursor must be closed.

If `SQLGetTypeInfo()` is called with an invalid *Data Type*, an empty result set is returned.

If either the `LONGDATACOMPAT` keyword or the `SQL_ATTR_LONGDATA_COMPAT` connection attribute is set, then `SQL_LONGVARBINARY`, `SQL_LONGVARCHAR` and `SQL_LONGVARGRAPHIC` will be returned for the *DATA\_TYPE* argument instead of `SQL_BLOB`, `SQL_CLOB` and `SQL_DBCLOB`.

The columns of the result set generated by this function are described below.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change. The data types returned are those that can be used in a `CREATE TABLE`, `ALTER TABLE`, `DDL` statement. Non-persistent data types such as the locator data types are not part of the returned result set. User-defined data types are not returned either.

### Columns returned by SQLGetTypeInfo

#### Column 1 TYPE\_NAME (VARCHAR(128) NOT NULL Data Type)

Data source-dependent data type name; for example, "CHAR()", "LONG VARBINARY". Applications must use this name in the `CREATE TABLE` and `ALTER TABLE` statements.

#### Column 2 DATA\_TYPE (SMALLINT NOT NULL Data Type)

SQL data type define values, for example, `SQL_VARCHAR`, `SQL_BLOB`, `SQL_DATE`, `SQL_INTEGER`.

#### Column 3 COLUMN\_SIZE (INTEGER Data Type)

If the data type is a character or binary string, then this column contains the maximum length in bytes; if it is a graphic (DBCS) string, this is the number of double byte characters for the column (the CLI/ODBC configuration keyword `Graphic` can change this default behaviour). If the data type is XML, zero is returned.

For date, time, timestamp data types, this is the total number of characters required to display the value when converted to character.

For numeric data types, this is the total number of digits (precision).

#### Column 4 LITERAL\_PREFIX (VARCHAR(128) Data Type)

Character that DB2 recognizes as a prefix for a literal of this data type. This column is null for data types where a literal prefix is not applicable.

#### Column 5 LITERAL\_SUFFIX (VARCHAR(128) Data Type)

Character that DB2 recognizes as a suffix for a literal of this data type. This column is null for data types where a literal prefix is not applicable.

#### Column 6 CREATE\_PARAMS (VARCHAR(128) Data Type)

The text of this column contains a list of keywords, separated by commas, corresponding to each parameter the application might specify in parenthesis when using the name in the `TYPE_NAME` column as a data

## SQLGetTypeInfo function (CLI) - Get data type information

type in SQL. The keywords in the list can be any of the following: LENGTH, PRECISION, SCALE. They appear in the order that the SQL syntax requires that they be used.

A NULL indicator is returned if there are no parameters for the data type definition, (such as INTEGER).

**Note:** The intent of CREATE\_PARAMS is to enable an application to customize the interface for a *DDL builder*. An application should expect, using this, only to be able to determine the number of arguments required to define the data type and to have localized text that could be used to label an edit control.

### Column 7 NULLABLE (SMALLINT NOT NULL Data Type)

Indicates whether the data type accepts a NULL value

- Set to SQL\_NO\_NULLS if NULL values are disallowed.
- Set to SQL\_NULLABLE if NULL values are allowed.
- Set to SQL\_NULLABLE\_UNKNOWN if it is not known whether NULL values are allowed or not.

### Column 8 CASE\_SENSITIVE (SMALLINT NOT NULL Data Type)

Indicates whether a character data type is case-sensitive in collations and comparisons. Valid values are SQL\_TRUE and SQL\_FALSE.

### Column 9 SEARCHABLE (SMALLINT NOT NULL Data Type)

Indicates how the data type is used in a WHERE clause. Valid values are:

- SQL\_UNSEARCHABLE : if the data type cannot be used in a WHERE clause.
- SQL\_LIKE\_ONLY : if the data type can be used in a WHERE clause only with the LIKE predicate.
- SQL\_ALL\_EXCEPT\_LIKE : if the data type can be used in a WHERE clause with all comparison operators except LIKE.
- SQL\_SEARCHABLE : if the data type can be used in a WHERE clause with any comparison operator.

### Column 10 UNSIGNED\_ATTRIBUTE (SMALLINT Data Type)

Indicates whether the data type is unsigned. The valid values are: SQL\_TRUE, SQL\_FALSE or NULL. A NULL indicator is returned if this attribute is not applicable to the data type.

### Column 11 FIXED\_PREC\_SCALE (SMALLINT NOT NULL Data Type)

Contains the value SQL\_TRUE if the data type is exact numeric and always has the same precision and scale; otherwise, it contains SQL\_FALSE.

### Column 12 AUTO\_INCREMENT (SMALLINT Data Type)

Contains SQL\_TRUE if a column of this data type is automatically set to a unique value when a row is inserted; otherwise, contains SQL\_FALSE.

### Column 13 LOCAL\_TYPE\_NAME (VARCHAR(128) Data Type)

This column contains any localized (native language) name for the data type that is different from the regular name of the data type. If there is no localized name, this column is NULL.

This column is intended for display only. The character set of the string is locale-dependent and is typically the default character set of the database.

### Column 14 MINIMUM\_SCALE (INTEGER Data Type)

The minimum scale of the SQL data type. If a data type has a fixed scale, the MINIMUM\_SCALE and MAXIMUM\_SCALE columns both contain the same value. NULL is returned where scale is not applicable.

## SQLGetTypeInfo function (CLI) - Get data type information

### Column 15 MAXIMUM\_SCALE (INTEGER Data Type)

The maximum scale of the SQL data type. NULL is returned where scale is not applicable. If the maximum scale is not defined separately in the DBMS, but is defined instead to be the same as the maximum length of the column, then this column contains the same value as the COLUMN\_SIZE column.

### Column 16 SQL\_DATA\_TYPE (SMALLINT NOT NULL Data Type)

The value of the SQL data type as it appears in the SQL\_DESC\_TYPE field of the descriptor. This column is the same as the DATA\_TYPE column (except for interval and datetime data types which CLI does not support).

### Column 17 SQL\_DATETIME\_SUB (SMALLINT Data Type)

This field is always NULL (CLI does not support interval and datetime data types).

### Column 18 NUM\_PREC\_RADIX (INTEGER Data Type)

If the data type is an approximate numeric type, this column contains the value 2 to indicate that COLUMN\_SIZE specifies a number of bits. For exact numeric types, this column contains the value 10 to indicate that COLUMN\_SIZE specifies a number of decimal digits. Otherwise, this column is NULL.

### Column 19 INTERVAL\_PRECISION (SMALLINT Data Type)

This field is always NULL (CLI does not support interval data types).

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 104. SQLGetTypeInfo SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle. <i>StatementHandle</i> had not been closed.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY004	SQL data type out of range.	An invalid <i>Data Type</i> was specified.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

**Example**

```
/* get data type information */
cliRC = SQLGetTypeInfo(hstmt, SQL_ALL_TYPES);
```

**SQLMoreResults function (CLI) - Determine if there are more result sets****Purpose**

<b>Specification:</b>	CLI 2.1	ODBC 1.0	
-----------------------	---------	----------	--

SQLMoreResults() determines whether there is more information available on the statement handle which has been associated with:

- an array input of parameter values for a query
- a stored procedure that is returning result sets
- or batched SQL

**Syntax**

```
SQLRETURN SQLMoreResults (SQLHSTMT StatementHandle); /* hstmt */
```

**Function arguments**

Table 105. SQLMoreResults arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle.

**Usage**

This function is used to return multiple results set in a sequential manner upon the execution of:

- a parameterized query with an array of input parameter values specified with the SQL\_ATTR\_PARAMSET\_SIZE statement attribute and SQLBindParameter(), or
- a stored procedure containing SQL queries, the cursors of which have been left open so that the result sets remain accessible when the stored procedure has finished execution. For this scenario, the stored procedure is typically trying to return multiple result sets.
- or batched SQL. When multiple SQL statements are batched together during a single SQLExecute() or SQLExecDirect().

After completely processing the first result set, the application can call SQLMoreResults() to determine if another result set is available. If the current result set has unfetched rows, SQLMoreResults() discards them by closing the cursor and, if another result set is available, returns SQL\_SUCCESS.

If all the result sets have been processed, SQLMoreResults() returns SQL\_NO\_DATA\_FOUND.

Applications that want to be able to manipulate more than one result set at the same time can use the CLI function SQLNextResult() to move a result set to another statement handle. SQLNextResult() does not support batched statements.

## SQLMoreResults function (CLI) - Determine if there are more result sets

When using batched SQL, `SQLExecute()` or `SQLExecDirect()` will only execute the first SQL statement in the batch. `SQLMoreResults()` can then be called to execute the next SQL statement and will return `SQL_SUCCESS` if the next statement is successfully executed. If there are no more statements to be executed, then `SQL_NO_DATA_FOUND` is returned. If the batched SQL statement is an `UPDATE`, `INSERT`, or `DELETE` statement, then `SQLRowCount()` can be called to determine the number of rows affected.

If `SQLCloseCursor()` or if `SQLFreeStmt()` is called with the `SQL_CLOSE` option, or `SQLFreeHandle()` is called with *HandleType* set to `SQL_HANDLE_STMT`, all pending result sets on this statement handle are discarded.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_STILL_EXECUTING`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

### Diagnostics

Table 106. *SQLMoreResults* SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.  The function was called while within a <code>BEGIN COMPOUND</code> and <code>END COMPOUND</code> SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

In addition `SQLMoreResults()` can return the `SQLSTATES` associated with `SQLExecute()`.

### Example

```
cliRC = SQLMoreResults(hstmt);
```



## SQLNativeSql function (CLI) - Get native SQL text

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

SQLNativeSql() is used to show how CLI interprets vendor escape clauses. If the original SQL string passed in by the application contained vendor escape clause sequences, then CLI will return the transformed SQL string that would be seen by the data source (with vendor escape clauses either converted or discarded, as appropriate).

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLNativeSqlW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLNativeSql (
    SQLHDBC          ConnectionHandle, /* hdbc */
    SQLCHAR          *InStatementText, /* szSqlStrIn */
    SQLINTEGER       TextLength1,     /* cbSqlStrIn */
    SQLCHAR          *OutStatementText, /* szSqlStr */
    SQLINTEGER       BufferLength,     /* cbSqlStrMax */
    SQLINTEGER       *TextLength2Ptr); /* pcbSqlStr */
```

### Function arguments

Table 107. SQLNativeSql arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection Handle
SQLCHAR *	<i>InStatementText</i>	input	Input SQL string
SQLINTEGER	<i>TextLength1</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>InStatementText</i> .
SQLCHAR *	<i>OutStatementText</i>	output	Pointer to buffer for the transformed output string
SQLINTEGER	<i>BufferLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>OutStatementText</i> .
SQLINTEGER *	<i>TextLength2Ptr</i>	output	The total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-terminator, available to return in <i>OutStatementText</i> . If the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) available to return is greater than or equal to <i>BufferLength</i> , the output SQL string in <i>OutStatementText</i> is truncated to <i>BufferLength</i> - 1 SQLCHAR or SQLWCHAR elements.

## SQLNativeSql function (CLI) - Get native SQL text

### Usage

This function is called when the application wishes to examine or display the transformed SQL string that would be passed to the data source by CLI. Translation (mapping) would only occur if the input SQL statement string contains vendor escape clause sequence(s).

CLI can only detect vendor escape clause syntax errors when `SQLNativeSql()` is called. Because CLI does not pass the transformed SQL string to the data source for preparation, syntax errors that are detected by the DBMS are not generated at this time. (The statement is not passed to the data source for preparation because the preparation may potentially cause the initiation of a transaction.)

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 108. SQLNativeSql SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated.	The buffer <i>OutStatementText</i> was not large enough to contain the entire SQL string, so truncation occurred. The argument <i>TextLength2Ptr</i> contains the total length of the untruncated SQL string. (Function returns with SQL_SUCCESS_WITH_INFO)
08003	Connection is closed.	The <i>ConnectionHandle</i> does not reference an open database connection.
37000	Invalid SQL syntax.	The input SQL string in <i>InStatementText</i> contained a syntax error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	The argument <i>InStatementText</i> is a NULL pointer. The argument <i>OutStatementText</i> is a NULL pointer.
HY090	Invalid string or buffer length.	The argument <i>TextLength1</i> was less than 0, but not equal to SQL_NTS. The argument <i>BufferLength</i> was less than 0.

### Restrictions

None.

## SQLNumParams function (CLI) - Get number of parameters in a SQL statement

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

## SQLNumParams function (CLI) - Get number of parameters in a SQL statement

SQLNumParams() returns the number of parameter markers in an SQL statement.

### Syntax

```
SQLRETURN SQLNumParams (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLSMALLINT *ParameterCountPtr); /* pcparr */
```

### Function arguments

Table 109. SQLNumParams arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT *	<i>ParameterCountPtr</i>	Output	Number of parameters in the statement.

### Usage

If the prepared SQL statement associated with *StatementHandle* contains batch SQL (multiple SQL statements separated by a semicolon ';'), the parameters are counted for the entire string and are not differentiated by the individual statements making up the batch.

This function can only be called after the statement associated with *StatementHandle* has been prepared. If the statement does not contain any parameter markers, *ParameterCountPtr* is set to 0.

An application can call this function to determine how many SQLBindParameter() (or SQLBindFileToParam()) calls are necessary for the SQL statement associated with the statement handle.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 110. SQLNumParams SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .

## SQLNumParams function (CLI) - Get number of parameters in a SQL statement

Table 110. SQLNumParams SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	This function was called before SQLPrepare() was called for the specified <i>StatementHandle</i>  The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

## SQLNextResult function (CLI) - Associate next result set with another statement handle

### Purpose

Specification:	CLI 7.x		
----------------	---------	--	--

SQLNextResult() allows non-sequential access to multiple result sets returned from a stored procedure.

### Syntax

```
SQLRETURN SQLNextResult (SQLHSTMT StatementHandle1  
                        SQLHSTMT StatementHandle2);
```

### Function arguments

Table 111. SQLNextResult arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle1</i>	input	Statement handle.
SQLHSTMT	<i>StatementHandle2</i>	input	Statement handle.

### Usage

A stored procedure returns multiple result sets by leaving one or more cursors open after exiting. The first result set is always accessed by using the statement handle that called the stored procedure. If multiple result sets are returned, either SQLMoreResults() or SQLNextResult() can be used to describe and fetch the result set.

SQLMoreResults() is used to close the cursor for the first result set and allow the next result set to be processed on the same statement handle, whereas

## SQLNextResult function (CLI) - Associate next result set with another statement handle

SQLNextResult() moves the next result set to *StatementHandle2*, without closing the cursor on *StatementHandle1*. Both functions return SQL\_NO\_DATA\_FOUND if there are no result sets to be fetched.

Using SQLNextResult() allows result sets to be processed in any order once they have been transferred to other statement handles. Mixed calls to SQLMoreResults() and SQLNextResult() are allowed until there are no more cursors (open result sets) on *StatementHandle1*.

When SQLNextResult() returns SQL\_SUCCESS, the next result set is no longer associated with *StatementHandle1*. Instead, the next result set is associated with *StatementHandle2*, as if a call to SQLExecDirect() had just successfully executed a query on *StatementHandle2*. The cursor, therefore, can be described using SQLNumResultCols(), SQLDescribeCol(), or SQLColAttribute().

After SQLNextResult() has been called, the result set now associated with *StatementHandle2* is removed from the chain of remaining result sets and cannot be used again in either SQLNextResult() or SQLMoreResults(). This means that for 'n' result sets, SQLNextResult() can be called successfully at most 'n-1' times.

If SQLCloseCursor() or if SQLFreeStmt() is called with the SQL\_CLOSE option, or SQLFreeHandle() is called with *HandleType* set to SQL\_HANDLE\_STMT, all pending result sets on this statement handle are discarded.

SQLNextResult() returns SQL\_ERROR if *StatementHandle2* has an open cursor or *StatementHandle1* and *StatementHandle2* are not on the same connection. If any errors or warnings are returned, SQLGetDiagRec() must always be called on *StatementHandle1*.

**Note:** SQLMoreResults() also works with a parameterized query with an array of input parameter values specified with the SQL\_ATTR\_ROW\_ARRAY\_SIZE statement attribute and SQLBindParameter(). SQLNextResult(), however, does not support this.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

### Diagnostics

Table 112. SQLNextResult SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication Link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate the memory required to support execution or completion of the function.

## SQLNextResult function (CLI) - Associate next result set with another statement handle

Table 112. SQLNextResult SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  <i>StatementHandle2</i> has an open cursor associated with it.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access the memory required to support execution or completion of the function.
HYT00	Time-out expired.	The time-out period expired before the data source returned the result set. The time-out period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

Only SQLMoreResults() can be used for parameterized queries and batched SQL.

### Example

```
/* use SQLNextResult to push Result Set 2 onto the second statement handle */  
cliRC = SQLNextResult(hstmt, hstmt2); /* open second cursor */
```

## SQLNumResultCols function (CLI) - Get number of result columns

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLNumResultCols() returns the number of columns in the result set associated with the input statement handle.

SQLPrepare() or SQLExecDirect() must be called before calling this function.

After calling this function, you can call SQLColAttribute(), or one of the bind column functions.

### Syntax

```
SQLRETURN SQLNumResultCols (  
    SQLHSTMT StatementHandle, /* hstmt */  
    SQLSMALLINT *ColumnCountPtr); /* pccol */
```

### Function arguments

Table 113. SQLNumResultCols arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLSMALLINT *	<i>ColumnCountPtr</i>	output	Number of columns in the result set

## SQLNumResultCols function (CLI) - Get number of result columns

### Usage

The function sets the output argument to zero if the last statement or function executed on the input statement handle did not generate a result set.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 114. SQLNumResultCols SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The function was called prior to calling <code>SQLPrepare()</code> or <code>SQLExecDirect()</code> for the <i>StatementHandle</i> .  The function was called while in a data-at-execute ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

### Authorization

None.

### Example

```
/* identify the number of output columns */  
cliRC = SQLNumResultCols(hstmt, &nResultCols);
```

## SQLParamData function (CLI) - Get next parameter for which a data value is needed

### Purpose

Specification:	CLI 2.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLParamData() is used in conjunction with SQLPutData() to send long data in pieces. It can also be used to send fixed-length data at execution time.

### Syntax

```
SQLRETURN SQLParamData (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLPOINTER       *ValuePtrPtr ); /* prgbValue */
```

### Function arguments

Table 115. SQLParamData arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle.
SQLPOINTER *	ValuePtrPtr	output	Pointer to a buffer in which to return the address of the <i>ParameterValuePtr</i> buffer specified in SQLBindParameter() (for parameter data) or the address of the <i>TargetValuePtr</i> buffer specified in SQLBindCol() (for column data), as contained in the SQL_DESC_DATA_PTR descriptor record field.
		input	Starting in Version 9.7 Fix Pack 1, when SQL_ATTR_INTERLEAVED_PUTDATA is set to TRUE, this is an input argument. The application provides a value for which it wants to put data in subsequent SQLPutData() calls.

### Usage

SQLParamData() returns SQL\_NEED\_DATA if there is at least one SQL\_DATA\_AT\_EXEC parameter for which data still has not been assigned. This function returns an application-provided value in ValuePtrPtr supplied by the application during a previous SQLBindParameter() call. SQLPutData() is called one or more times (in the case of long data) to send the parameter data. SQLParamData() is called to signal that all the data has been sent for the current parameter and to advance to the next SQL\_DATA\_AT\_EXEC parameter. SQL\_SUCCESS is returned when all the parameters have been assigned data values and the associated statement has been executed successfully. If any errors occur during or before actual statement execution, SQL\_ERROR is returned.

If SQLParamData() returns SQL\_NEED\_DATA, then only SQLPutData() or SQLCancel() calls can be made. All other function calls using this statement handle will fail. In addition, all function calls referencing the parent connection handle of StatementHandle will fail if they involve changing any attribute or state of that connection; that is, that following function calls on the parent connection handle are also not permitted:

- SQLSetConnectAttr()
- SQLEndTran()



## SQLParamData function (CLI) - Get next parameter for which a data value is needed

However, calls to the `SQLEndTran()` function specifying `SQL_ROLLBACK` as completion type are allowed when the `SQL_ATTR_FORCE_ROLLBACK` connection attribute is set, the `StreamPutData` configuration keyword is set to 1, and autocommit mode is enabled.

Should they be invoked during an `SQL_NEED_DATA` sequence, these functions will return `SQL_ERROR` with `SQLSTATE` of `HY010` and the processing of the `SQL_DATA_AT_EXEC` parameters will not be affected.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_NEED_DATA`
- `SQL_STILL_EXECUTING`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NEED_DATA`

### Diagnostics

`SQLParamData()` can return any `SQLSTATE` returned by the `SQLPrepare()`, `SQLExecDirect()`, and `SQLExecute()` functions. In addition, the following diagnostics can also be generated:

Table 116. *SQLParamData SQLSTATES*

SQLSTATE	Description	Explanation
07006	Invalid conversion.	Transfer of data between CLI and the application variables would result in incompatible data conversion.
22026	String data, length mismatch	<p>The <code>SQL_NEED_LONG_DATA_LEN</code> information type in <code>SQLGetInfo()</code> was 'Y' and less data was sent for a long parameter (the data type was <code>SQL_LONGVARCHAR</code>, <code>SQL_LONGVARBINARY</code>, or other long data type) than was specified with the <i>StrLen_or_IndPtr</i> argument in <code>SQLBindParameter()</code>.</p> <p>The <code>SQL_NEED_LONG_DATA_LEN</code> information type in <code>SQLGetInfo()</code> was 'Y' and less data was sent for a long column (the data type was <code>SQL_LONGVARCHAR</code>, <code>SQL_LONGVARBINARY</code>, or other long data type) than was specified in the length buffer corresponding to a column in a row of data that was updated with <code>SQLSetPos()</code>.</p>
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY000	General error.	An error occurred for which there was no specific <code>SQLSTATE</code> and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <code>SQLGetDiagRec()</code> in the argument <i>MessageText</i> describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

## SQLParamData function (CLI) - Get next parameter for which a data value is needed

Table 116. SQLParamData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<code>SQLParamData()</code> was called out of sequence. This call is only valid after an <code>SQLExecDirect()</code> or an <code>SQLExecute()</code> , or after an <code>SQLPutData()</code> call.  Even though this function was called after an <code>SQLExecDirect()</code> or an <code>SQLExecute()</code> call, there were no <code>SQL_DATA_AT_EXEC</code> parameters (left) to process.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY092	Option type out of range.	The <i>FileOptions</i> argument of a previous <code>SQLBindFileToParam()</code> operation was not valid.
HY506	Error closing a file.	Error encountered while trying to close a temporary file.
HY509	Error deleting a file.	Error encountered while trying to delete a temporary file.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

### Restrictions

None.

### Example

```
/* get next parameter for which a data value is needed */  
cliRC = SQLParamData(hstmt, (SQLPOINTER *)&valuePtr);
```

## SQLParamOptions function (CLI) - Specify an input array for a parameter

### Deprecated

#### Note:

In ODBC 3.0, `SQLParamOptions()` has been deprecated and replaced with `SQLSetStmtAttr()`.

Although this version of CLI continues to support `SQLParamOptions()`, use `SQLSetStmtAttr()` in your CLI programs so that they conform to the latest standards.

### Migrating to the new function

The statement:

```
SQLParamOptions(hstmt, crow, pirow);
```

for example, would be rewritten using the new function as:

```
SQLSetStmtAttr(hstmt, fOption, pvParam, fStrLen);
```

## SQLPrepare function (CLI) - Prepare a statement

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

SQLPrepare() associates an SQL statement or XQuery expression with the input statement handle provided. The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) or a colon followed by a name (:name) into the SQL string at the appropriate position. The application can reference this prepared statement by passing the statement handle to other functions.

**Note:** For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the XMLQUERY function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in XMLQUERY for execution.

If the statement handle has been previously used with a query statement (or any function that returns a result set), either SQLCloseCursor() or SQLFreeStmt() with the SQL\_CLOSE option must be called to close the cursor before calling SQLPrepare().

XQuery expressions must be prefixed with the "XQUERY" keyword. To prepare and execute XQuery expressions without having to include this keyword, set the statement attribute SQL\_ATTR\_XQUERY\_STATEMENT to SQL\_TRUE before calling SQLPrepare() or SQLExecDirect().

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLPrepareW(). Refer to "Unicode functions (CLI)" on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLPrepare (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *StatementText, /* szSqlStr */
    SQLINTEGER TextLength); /* cbSqlStr */
```

### Function arguments

Table 117. SQLPrepare arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle. There must not be an open cursor associated with StatementHandle.
SQLCHAR *	StatementText	input	SQL statement string
SQLINTEGER	TextLength	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the StatementText argument, or SQL_NTS if StatementText is null-terminated.

## SQLPrepare function (CLI) - Prepare a statement

### Usage

Deferred prepare is on by default. The prepare request is not sent to the server until either `SQLDescribeParam()`, `SQLExecute()`, `SQLNumResultCols()`, `SQLDescribeCol()`, or `SQLColAttribute()` is called using the same statement handle as the prepared statement. This minimizes network flow and improves performance.

If the SQL statement text contains vendor escape clause sequences, CLI will first modify the SQL statement text to the appropriate DB2 specific format before submitting it to the database for preparation. If the application does not generate SQL statements that contain vendor escape clause sequences then the `SQL_ATTR_NOSCAN` statement attribute should be set to `SQL_NOSCAN` at the connection level so that CLI does not perform a scan for any vendor escape clauses.

Once a statement has been prepared using `SQLPrepare()`, the application can request information about the format of the result set (if the statement was a query) by calling:

- `SQLNumResultCols()`
- `SQLDescribeCol()`
- `SQLColAttribute()`

Information about the parameter markers in *StatementText* can be requested using the following:

- `SQLDescribeParam()`
- `SQLNumParams()`

**Note:** The first invocation of any of the above functions except `SQLNumParams()` will force the PREPARE request to be sent to the server if deferred prepare is enabled.

The SQL statement string might contain parameter markers and `SQLNumParams()` can be called to determine the number of parameter markers in the statement. A parameter marker is represented by a “?” character or a colon followed by a name (*:name*), and is used to indicate a position in the statement where an application-supplied value is to be substituted when `SQLExecute()` is called. The bind parameter functions, `SQLBindParameter()`, `SQLSetParam()` and `SQLBindFileToParam()`, are used to bind or associate application variables with each parameter marker and to indicate if any data conversion should be performed at the time the data is transferred. An application can call `SQLDescribeParam()` to retrieve information about the data expected by the database server for the parameter marker.

All parameters must be bound before calling `SQLExecute()`.

Refer to the PREPARE statement for information on rules related to parameter markers.

Once the application has processed the results from the `SQLExecute()` call, it can execute the statement again with new (or the same) parameter values.

The SQL statement can be COMMIT or ROLLBACK and executing either of these statements has the same effect as calling `SQLEndTran()` on the current connection handle.

## SQLPrepare function (CLI) - Prepare a statement

If the SQL statement is a positioned DELETE or a positioned UPDATE, the cursor referenced by the statement must be defined on a separate statement handle under the same connection handle and same isolation level.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 118. SQLPrepare SQLSTATEs

SQLSTATE	Description	Explanation
01504	The UPDATE or DELETE statement does not include a WHERE clause.	<i>StatementText</i> contained an UPDATE or DELETE statement which did not contain a WHERE clause.
01508	Statement disqualified for blocking.	The statement was disqualified for blocking for reasons other than storage.
21S01	Insert value list does not match column list.	<i>StatementText</i> contained an INSERT statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degrees of derived table does not match column list.	<i>StatementText</i> contained a CREATE VIEW statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22018	Invalid character value for cast specification.	<i>StatementText</i> contained an SQL statement that contained a literal or parameter and the value was incompatible with the data type of the associated table column.
22019	Invalid escape character	The argument <i>StatementText</i> contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	The argument <i>StatementText</i> contained "LIKE <i>pattern value</i> ESCAPE <i>escape character</i> " in the WHERE clause, and the character following the escape character in the pattern value was not one of "%" or "_".
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
34000	Invalid cursor name.	<i>StatementText</i> contained a positioned DELETE or a positioned UPDATE and the cursor referenced by the statement being executed was not open.
37xxx <sup>a</sup>	Invalid SQL syntax.	<i>StatementText</i> contained one or more of the following: <ul style="list-style-type: none"><li>• an SQL statement that the connected database server could not prepare</li><li>• a statement containing a syntax error</li></ul>
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to deadlock or timeout.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.

## SQLPrepare function (CLI) - Prepare a statement

Table 118. SQLPrepare SQLSTATES (continued)

SQLSTATE	Description	Explanation
42xxx <sup>a</sup>	Syntax Error or Access Rule Violation.	425xx indicates the authorization ID does not have permission to execute the SQL statement contained in <i>StatementText</i> .  Other 42xxx SQLSTATES indicate a variety of syntax or access problems with the statement.
58004	Unexpected system failure.	Unrecoverable system error.
S0001	Database object already exists.	<i>StatementText</i> contained a CREATE TABLE or CREATE VIEW statement and the table name or view name specified already existed.
S0002	Database object does not exist.	<i>StatementText</i> contained an SQL statement that references a table name or a view name which did not exist.
S0011	Index already exists.	<i>StatementText</i> contained a CREATE INDEX statement and the specified index name already existed.
S0012	Index not found.	<i>StatementText</i> contained a DROP INDEX statement and the specified index name did not exist.
S0021	Column already exists.	<i>StatementText</i> contained an ALTER TABLE statement and the column specified in the ADD clause was not unique or identified an existing column in the base table.
S0022	Column not found.	<i>StatementText</i> contained an SQL statement that references a column name which did not exist.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel () was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	<i>StatementText</i> was a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData (), SQLPutData ()) operation.  The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The argument <i>TextLength</i> was less than 1, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

**Note:**

**a** xxx refers to any SQLSTATE with that class code. Example, 37xxx refers to any SQLSTATE in the 37 class.

## SQLPrepare function (CLI) - Prepare a statement

**Note:** Not all DBMSs report all of the above diagnostic messages at prepare time. If deferred prepare is left on as the default behavior (controlled by the `SQL_ATTR_DEFERRED_PREPARE` statement attribute), then these errors could occur when the PREPARE is flowed to the server. The application must be able to handle these conditions when calling functions that cause this flow. These functions include `SQLExecute()`, `SQLDescribeParam()`, `SQLNumResultCols()`, `SQLDescribeCol()`, and `SQLColAttribute()`.

### Authorization

None.

### Example

```
SQLCHAR *stmt = (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? ";

/* ... */

/* prepare the statement */
cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);
```

---

## SQLPrimaryKeys function (CLI) - Get primary key columns of a table

The `SQLPrimaryKeys()` function returns a list of column names that comprise the primary key for a table. The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLPrimaryKeysW()`. For information about ANSI to Unicode function mappings, see "Unicode functions (CLI)" on page 5.

### Syntax

```
SQLRETURN SQLPrimaryKeys (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLCHAR           *CatalogName,   /* szCatalogName */
    SQLSMALLINT       NameLength1,    /* cbCatalogName */
    SQLCHAR           *SchemaName,    /* szSchemaName */
    SQLSMALLINT       NameLength2,    /* cbSchemaName */
    SQLCHAR           *TableName,     /* szTableName */
    SQLSMALLINT       NameLength3);  /* cbTableName */
```

### Function arguments

Table 119. `SQLPrimaryKeys` arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and <code>SQL_SUCCESS</code> is returned. Otherwise, this is a valid filter for DBMSs that supports 3-part naming.

## SQLPrimaryKeys function (CLI) - Get primary key columns of a table

Table 119. SQLPrimaryKeys arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or SQL_NTS if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	The schema qualifier of table name.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	The table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableName</i> , or SQL_NTS if <i>TableName</i> is null-terminated.

### Usage

The SQLPrimaryKeys() function returns the primary key columns from a single table. You cannot use search patterns to specify any of the arguments.

The result set contains the columns that are listed in Columns Returned By SQLPrimaryKeys, ordered by TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME and ORDINAL\_POSITION columns.

In many cases, calls to the SQLPrimaryKeys() function map to a complex and thus expensive query against the system catalog, so you should use these calls sparingly, and save the results rather than repeating calls.

If the schema name is not provided, the schema name defaults to the name that is in effect for the current connection.

Call SQLGetInfo() with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

In Version 9.7 Fix Pack 5, you can specify \*ALL as a value in the *SchemaName* to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns Returned By SQLPrimaryKeys

#### Column 1 TABLE\_CAT (VARCHAR(128))

The primary key table catalog name. The value is NULL if this table does not have catalogs.



## SQLPrimaryKeys function (CLI) - Get primary key columns of a table

**Column 2** TABLE\_SCHEM (VARCHAR(128))

The name of the schema that contains TABLE\_NAME.

**Column 3** TABLE\_NAME (VARCHAR(128) not NULL)

The name of the specified table.

**Column 4** COLUMN\_NAME (VARCHAR(128) not NULL)

The primary key column name.

**Column 5** KEY\_SEQ (SMALLINT not NULL)

The column sequence number in the primary key, starting with 1.

**Column 6** PK\_NAME (VARCHAR(128))

The primary key identifier. NULL if not applicable to the data source.

**Note:** The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the SQLPrimaryKeys() result set in ODBC.

If the specified table does not contain a primary key, an empty result set is returned.

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 120. SQLPrimaryKeys SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <i>SQLCancel()</i> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The function was called while in a data-at-execute ( <i>SQLParamData()</i> , <i>SQLPutData()</i> ) operation.  The function was called while in a BEGIN COMPOUND and END COMPOUND SQL operation.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name-length arguments was less than 0, but not equal to <i>SQL_NTS</i> .
HYC00	Driver not capable.	CLI does not support <i>catalog</i> as a qualifier for table name.

## SQLPrimaryKeys function (CLI) - Get primary key columns of a table

Table 120. SQLPrimaryKeys SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* get the primary key columns of a table */  
cliRC = SQLPrimaryKeys(hstmt, NULL, 0, tbSchema, SQL_NTS, tbName, SQL_NTS);
```

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

The SQLProcedureColumns() function returns a list of input and output parameters that are associated with a stored procedure.

The information is returned in an SQL result set, which you can retrieve using the same functions that you use to process a result set that is generated by a query.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is SQLProcedureColumnsW(). For information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLProcedureColumns(  
    SQLHSTMT          StatementHandle, /* hstmt */  
    SQLCHAR           *CatalogName,   /* szProcCatalog */  
    SQLSMALLINT       NameLength1,    /* cbProcCatalog */  
    SQLCHAR           *SchemaName,    /* szProcSchema */  
    SQLSMALLINT       NameLength2,    /* cbProcSchema */  
    SQLCHAR           *ProcName,      /* szProcName */  
    SQLSMALLINT       NameLength3,    /* cbProcName */  
    SQLCHAR           *ColumnName,    /* szColumnName */  
    SQLSMALLINT       NameLength4);  /* cbColumnName */
```

### Function arguments

Table 121. SQLProcedureColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	Input	The statement handle.

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

Table 121. SQLProcedureColumns arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS is returned. Otherwise, this is a valid filter for DBMSs that supports 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or SQL_NTS if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by schema name.  For DB2 for z/OS, the stored procedures are in one schema; the only acceptable value for the <i>SchemaName</i> argument is a null pointer. If a value is specified, an empty result set and SQL_SUCCESS are returned. For DB2 Database for Linux, UNIX, and Windows, <i>SchemaName</i> can contain a valid pattern value. For more information about valid search patterns, see the catalog functions input arguments.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>ProcName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by the procedure name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ProcName</i> , or SQL_NTS if <i>ProcName</i> is null-terminated.
SQLCHAR *	<i>ColumnName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by the parameter name. Use this argument to further qualify the result set that is already restricted by specifying a non-empty value for <i>ProcName</i> , <i>SchemaName</i> , or both.
SQLSMALLINT	<i>NameLength4</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ColumnName</i> , or SQL_NTS if <i>ColumnName</i> is null-terminated.

### Usage

The SQLProcedureColumns() function returns the information in a result set, ordered by PROCEDURE\_CAT, PROCEDURE\_SCHEM, PROCEDURE\_NAME, and COLUMN\_TYPE. Columns returned by SQLProcedureColumns lists the columns in the result set. Applications should be aware that columns that are beyond the last column might be defined in future releases.

In many cases, calls to the SQLProcedureColumns() function map to a complex and thus expensive query against the system catalog, so you should use the calls sparingly, and save the results rather than repeating calls.

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

Call `SQLGetInfo()` with the `SQL_MAX_CATALOG_NAME_LEN`, `SQL_MAX_SCHEMA_NAME_LEN`, and `SQL_MAX_COLUMN_NAME_LEN` to determine respectively the actual lengths of the `TABLE_CAT`, `TABLE_SCHEM`, and `COLUMN_NAME` columns that are supported by the connected DBMS.

If the `SQL_ATTR_LONGDATA_COMPAT` connection attribute is set, LOB column types are reported as `LONG VARCHAR`, `LONG VARBINARY` or `LONG VARGRAPHIC` types.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

If the stored procedure is at a DB2 for z/OS server, the name of the stored procedure must be registered in the `SYSIBM.SYSPROCEDURES` catalog table of the server. For V8 and later servers, the stored procedures must be registered in the `SYSIBM.SYSROUTINES` and `SYSIBM.SYSPARAMS` catalog tables of the server.

For versions of other DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set is returned.

CLI returns information about the input, input/output, and output parameters that are associated with the stored procedure, but cannot return descriptor information for any result sets that the stored procedure might return.

In Version 9.7 Fix Pack 5, you can specify `*ALL` as a value in the `SchemaName` to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not required to specify `*ALL`, as this behavior is the default in CLI. Alternatively, you can set the `SchemaFilter` IBM Data Server Driver configuration keyword or the `Schema List` CLI/ODBC configuration keyword to `*ALL`.

### Columns returned by SQLProcedureColumns

#### Column 1 PROCEDURE\_CAT (VARCHAR(128))

The procedure catalog name. The value is NULL if this procedure does not have catalogs.

#### Column 2 PROCEDURE\_SCHEM (VARCHAR(128))

The name of the schema that contains `PROCEDURE_NAME`. This is NULL for DB2 for z/OS `SQLProcedureColumns()` result sets.

#### Column 3 PROCEDURE\_NAME (VARCHAR(128))

The name of the procedure.

#### Column 4 COLUMN\_NAME (VARCHAR(128))

The name of the parameter.

#### Column 5 COLUMN\_TYPE (SMALLINT not NULL)

Identifies the type of information that is associated with this row. The values that can be returned are:

- `SQL_PARAM_INPUT` is an input parameter.
- `SQL_PARAM_INPUT_OUTPUT` is an input / output parameter.
- `SQL_PARAM_OUTPUT` is an output parameter.

The values which are defined in the ODBC specification but are not returned:

- `SQL_PARAM_TYPE_UNKNOWN` : the parameter type is unknown.

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

- SQL\_RETURN\_VALUE is the return value of the procedure, in the procedure column.
- SQL\_RESULT\_COL is a column in the result set.

### Column 6 DATA\_TYPE (SMALLINT not NULL)

The SQL data type.

### Column 7 TYPE\_NAME (VARCHAR(128) not NULL)

The character string that represents the name of the data type that corresponds to DATA\_TYPE.

### Column 8 COLUMN\_SIZE (INTEGER)

For XML arguments in SQL routines, zero is returned (as XML arguments have no length). For cataloged external routines, however, XML parameters are declared as XML AS CLOB(n), in which case COLUMN\_SIZE is the cataloged length, n.

If the DATA\_TYPE column value denotes a character or binary string, this column contains the maximum length in SQLCHAR or SQLWCHAR elements. If DATA\_TYPE column value is a graphic (DBCS) string, the COLUMN\_SIZE is the number of double byte SQLCHAR or SQLWCHAR elements for the parameter.

For date, time, and timestamp data types, this is the total number of SQLCHAR or SQLWCHAR elements that are required to display the value when converted to character data type.

For numeric data types, COLUMN\_SIZE value is either the total number of digits or the total number of bits that are allowed in the column, depending on the value in the NUM\_PREC\_RADIX column in the result set.

See the table of data type precision.

### Column 9 BUFFER\_LENGTH (INTEGER)

The maximum number of bytes for the associated C buffer to store data from this parameter if SQL\_C\_DEFAULT is specified on the SQLBindCol(), SQLGetData() and SQLBindParameter() calls. This length excludes any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

For XML arguments in SQL routines, zero is returned (as XML arguments have no length). For cataloged external routines, however, XML parameters are declared as XML AS CLOB(n), in which case BUFFER\_LENGTH is the cataloged length, n.

See the table of data type length.

### Column 10 DECIMAL\_DIGITS (SMALLINT)

The scale of the parameter. NULL is returned for data types where scale is not applicable.

See the table of data type scale.

### Column 11 NUM\_PREC\_RADIX (SMALLINT)

Either 10, 2, or NULL. If DATA\_TYPE is an approximate numeric data type, this column contains the value 2, and the COLUMN\_SIZE column contains the number of bits that are allowed in the parameter.

If DATA\_TYPE is an exact numeric data type, this column contains the value 10, and the COLUMN\_SIZE and DECIMAL\_DIGITS columns contain the number of decimal digits that are allowed for the parameter.

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

For numeric data types, the DBMS can return a NUM\_PREC\_RADIX of either 10 or 2.

NULL is returned for data types where radix is not applicable.

### Column 12 NULLABLE (SMALLINT not NULL)

SQL\_NO\_NULLS if the parameter does not accept NULL values.

SQL\_NULLABLE if the parameter accepts NULL values.

### Column 13 REMARKS (VARCHAR(254))

Might contain descriptive information about the parameter.

### Column 14 COLUMN\_DEF (VARCHAR)

The default value of the column.

If NULL was specified as the default value, this column is the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, this column contains TRUNCATED, not enclosed in single quotation marks. If no default value is specified, this column is NULL.

You can use the value of COLUMN\_DEF to generate a new column definition, except when COLUMN\_DEF contains the value TRUNCATED.

### Column 15 SQL\_DATA\_TYPE (SMALLINT not NULL)

The value of the SQL data type as it is displayed in the SQL\_DESC\_TYPE field of the descriptor. This column is the same as the DATA\_TYPE column except for datetime data types (CLI does not support interval data types).

For datetime data types, the SQL\_DATA\_TYPE field in the result set is SQL\_DATETIME, and the SQL\_DATETIME\_SUB field returns the subcode for the specific datetime data type (SQL\_CODE\_DATE, SQL\_CODE\_TIME or SQL\_CODE\_TIMESTAMP).

### Column 16 SQL\_DATETIME\_SUB (SMALLINT)

The subtype code for datetime data types. For all other data types this column returns a NULL value (including interval data types that CLI does not support).

### Column 17 CHAR\_OCTET\_LENGTH (INTEGER)

The maximum length in bytes of a character data type column. For all other data types, this column returns a NULL.

### Column 18 ORDINAL\_POSITION (INTEGER NOT NULL)

Contains the ordinal position of the parameter that is given by COLUMN\_NAME in this result set. This is the ordinal position of the argument to be provided on the CALL statement. The leftmost argument has an ordinal position of 1.

### Column 19 IS\_NULLABLE (Varchar)

- "NO" if the column cannot contain NULLs.
- "YES" if the column can include NULLs.
- Zero-length string if nullability is unknown.

ISO rules are followed to determine nullability.

An ISO SQL-compliant DBMS cannot return an empty string.

The value that is returned for this column is different than the value that is returned for the NULLABLE column. (See the description of the NULLABLE column.)

### Note:

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

- The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those defined for the SQLProcedureColumns() result set in ODBC.
- If two modules contain procedures that share the same name, SQLProcedureColumns() returns details about both procedures.

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 122. SQLProcedureColumns SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42601	PARMLIST syntax error.	The PARMLIST value in the stored procedures catalog table contains a syntax error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.  The function was called while in a BEGIN COMPOUND and END COMPOUND SQL operation.  An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.  The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

## SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

### Restrictions

The SQLProcedureColumns() function does not return information about the attributes of result sets that might be returned from stored procedures.

If an application is connected to a DB2 server that does not provide support for a stored procedure catalog, or does not provide support for stored procedures, the SQLProcedureColumns() function returns an empty result set.

### Example

```
/* get input/output parameter information for a procedure */
sqlrc = SQLProcedureColumns(hstmt,
                            NULL,
                            0, /* catalog name not used */
                            (unsigned char *)colSchemaNamePattern,
                            SQL_NTS, /* schema name not currently used */
                            (unsigned char *)procname,
                            SQL_NTS,
                            colNamePattern,
                            SQL_NTS); /* all columns */
```

---

## SQLProcedures function (CLI) - Get list of procedure names

The SQLProcedures() function returns a list of stored procedure names that have been registered at the server, and which match the specified search pattern.

The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is SQLProceduresW(). For information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLProcedures (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *CatalogName, /* szProcCatalog */
    SQLSMALLINT NameLength1, /* cbProcCatalog */
    SQLCHAR *SchemaName, /* szProcSchema */
    SQLSMALLINT NameLength2, /* cbProcSchema */
    SQLCHAR *ProcName, /* szProcName */
    SQLSMALLINT NameLength3); /* cbProcName */
```

### Function arguments

Table 123. SQLProcedures arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	Input	The statement handle.



## SQLProcedures function (CLI) - Get list of procedure names

Table 123. SQLProcedures arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS is returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or SQL_NTS if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by schema name.  For DB2 for z/OS, the stored procedures are in one schema; the only acceptable value for the <i>SchemaName</i> argument is a null pointer. If a value is specified, an empty result set and SQL_SUCCESS are returned. For DB2 Database for Linux, UNIX, and Windows, <i>SchemaName</i> can contain a valid pattern value. For more information about valid search patterns, see the catalog functions input arguments.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>ProcName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>ProcName</i> , or SQL_NTS if <i>ProcName</i> is null-terminated.

### Usage

The result set that is returned by the SQLProcedures() function contains the columns that are listed in Columns returned by SQLProcedures in the order given. The rows are ordered by PROCEDURE\_CAT, PROCEDURE\_SCHEMA, and PROCEDURE\_NAME.

In many cases, calls to the SQLProcedures() function map to a complex and thus expensive query against the system catalog, so you should use them sparingly, and save the results rather than repeating calls.

Call SQLGetInfo() with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

If the SQL\_ATTR\_LONGDATA\_COMPAT connection attribute is set, LOB column types are reported as LONG VARCHAR, LONG VARBINARY, or LONG VARGRAPHIC types.

## SQLProcedures function (CLI) - Get list of procedure names

If the stored procedure is at a DB2 for z/OS server, the name of the stored procedures must be registered in the server's SYSIBM.SYSPROCEDURES catalog table. For V8 and later servers, the stored procedure must be registered in the server's SYSIBM.SYSROUTINES and SYSIBM.SYSPARAMS catalog tables.

For other versions of DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set is returned.

In Version 9.7 Fix Pack 5, you can specify \*ALL as a value in the *SchemaName* to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns returned by SQLProcedures

#### Column 1 PROCEDURE\_CAT (VARCHAR(128))

The procedure catalog name. The value is NULL if this procedure does not have catalogs.

#### Column 2 PROCEDURE\_SCHEM (VARCHAR(128))

The name of the schema that contains PROCEDURE\_NAME.

#### Column 3 PROCEDURE\_NAME (VARCHAR(128) NOT NULL)

The name of the procedure.

#### Column 4 NUM\_INPUT\_PARAMS (INTEGER not NULL)

The number of input parameters. INOUT parameters are not counted as part of this number.

To determine information regarding INOUT parameters, examine the COLUMN\_TYPE column that is returned by SQLProcedureColumns().

#### Column 5 NUM\_OUTPUT\_PARAMS (INTEGER not NULL)

The number of output parameters. INOUT parameters are not counted as part of this number.

To determine information regarding INOUT parameters, examine the COLUMN\_TYPE column that is returned by SQLProcedureColumns().

#### Column 6 NUM\_RESULT\_SETS (INTEGER not NULL)

The number of result sets that are returned by the procedure.

You should not use this column, it is reserved for future use by ODBC.

#### Column 7 REMARKS (VARCHAR(254))

Contains the descriptive information about the procedure.

#### Column 8 PROCEDURE\_TYPE (SMALLINT)

Defines the procedure type:

- SQL\_PT\_UNKNOWN: It cannot be determined whether the procedure returns a value.
- SQL\_PT\_PROCEDURE: The returned object is a procedure that does not have a return value.
- SQL\_PT\_FUNCTION: The returned objects is a function that has a return value

CLI always returns SQL\_PT\_PROCEDURE.

## SQLProcedures function (CLI) - Get list of procedure names

### Note:

- The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those that are defined for the SQLProcedures() result set in ODBC.
- If two modules contain procedures that share the same name, the SQLProcedures() function returns details about both procedures.

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 124. SQLProcedures SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. The function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

## SQLProcedures function (CLI) - Get list of procedure names

### Restrictions

If an application is connected to a DB2 server that does not provide support for a stored procedure catalog, or does not provide support for stored procedures, `SQLProcedureColumns()` will return an empty result set.

---

## SQLPutData function (CLI) - Passing data value for a parameter

### Purpose

Specification:	CLI 2.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

`SQLPutData()` is called following an `SQLParamData()` call returning `SQL_NEED_DATA` to supply parameter data values. This function can be used to send large parameter values in pieces.

### Syntax

```
SQLRETURN SQLPutData (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLPOINTER DataPtr, /* rgbValue */
    SQLLEN StrLen_or_Ind); /* cbValue */
```

### Function arguments

Table 125. `SQLPutData` arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLPOINTER	<i>DataPtr</i>	Input	Pointer to the actual data, or portion of data, for a parameter. The data must be in the form specified in the <code>SQLBindParameter()</code> call that the application used when specifying the parameter.
SQLLEN	<i>StrLen_or_Ind</i>	Input	The length of <i>DataPtr</i> . Specifies the amount of data sent in a call to <code>SQLPutData()</code> .  The amount of data can vary with each call for a given parameter. The application can also specify <code>SQL_NTS</code> or <code>SQL_NULL_DATA</code> for <i>StrLen_or_Ind</i> .  <i>StrLen_or_Ind</i> is ignored for all fixed length C buffer types, such as date, time, timestamp, and all numeric C buffer types.  For cases where the C buffer type is <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> , or if <code>SQL_C_DEFAULT</code> is specified as the C buffer type and the C buffer type default is <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> , this is the number of bytes of data in the <i>DataPtr</i> buffer.

### Usage

The application calls `SQLPutData()` after calling `SQLParamData()` on a statement in the `SQL_NEED_DATA` state to supply the data values for an `SQL_DATA_AT_EXEC` parameter. Long data can be sent in pieces via repeated calls to `SQLPutData()`. CLI generates a temporary file for each `SQL_DATA_AT_EXEC` parameter to which each piece of data is appended when `SQLPutData()` is called. The path in which CLI creates its temporary files can be set

## SQLPutData function (CLI) - Passing data value for a parameter

using the `TEMPDIR` keyword in the `db2cli.ini` file. If this keyword is not set, CLI attempts to write to the path specified by the environment variables `TEMP` or `TMP`. After all the pieces of data for the parameter have been sent, the application calls `SQLParamData()` again to proceed to the next `SQL_DATA_AT_EXEC` parameter, or, if all parameters have data values, to execute the statement.

`SQLPutData()` cannot be called more than once for a fixed length C buffer type, such as `SQL_C_LONG`.

After an `SQLPutData()` call, the only legal function calls are `SQLParamData()`, `SQLCancel()`, or another `SQLPutData()` if the input data is character or binary data. As with `SQLParamData()`, all other function calls using this statement handle will fail. In addition, all function calls referencing the parent connection handle of *StatementHandle* will fail if they involve changing any attribute or state of that connection; that is, the following function calls on the parent connection handle are also not permitted:

- `SQLSetConnectAttr()`
- `SQLEndTran()`

However, calls to the `SQLEndTran()` function specifying `SQL_ROLLBACK` as completion type are allowed when the `SQL_ATTR_FORCE_ROLLBACK` connection attribute is set, the `StreamPutData` configuration keyword is set to 1, and autocommit mode is enabled.

Should they be invoked during an `SQL_NEED_DATA` sequence, these functions will return `SQL_ERROR` with `SQLSTATE` of `HY010` and the processing of the `SQL_DATA_AT_EXEC` parameters will not be affected.

If one or more calls to `SQLPutData()` for a single parameter results in `SQL_SUCCESS`, attempting to call `SQLPutData()` with `StrLen_or_Ind` set to `SQL_NULL_DATA` for the same parameter results in an error with `SQLSTATE` of `22005`. This error does not result in a change of state; the statement handle is still in a *Need Data* state and the application can continue sending parameter data.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_STILL_EXECUTING`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Some of the following diagnostics conditions might also be reported on the final `SQLParamData()` call rather than at the time the `SQLPutData()` is called.

Table 126. *SQLPutData SQLSTATES*

SQLSTATE	Description	Explanation
01004	Data truncated.	The data sent for a numeric parameter was truncated without the loss of significant digits.
		Timestamp data sent for a date or time column was truncated.
		Function returns with <code>SQL_SUCCESS_WITH_INFO</code> .
22001	String data right truncation.	More data was sent for a binary or char data than the data source can support for that column.

## SQLPutData function (CLI) - Passing data value for a parameter

Table 126. SQLPutData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
22003	Numeric value out of range.	The data sent for a numeric parameter caused the whole part of the number to be truncated when assigned to the associated column.  SQLPutData() was called more than once for a fixed length parameter.
22005	Error in assignment.	The data sent for a parameter was incompatible with the data type of the associated table column.
22007	Invalid datetime format.	The data value sent for a date, time, or timestamp parameters was invalid.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	The argument <i>DataPtr</i> was a NULL pointer, and the argument <i>StrLen_or_Ind</i> was neither 0 nor SQL_NULL_DATA.
HY010	Function sequence error.	The statement handle <i>StatementHandle</i> must be in a need data state and must have been positioned on an SQL_DATA_AT_EXEC parameter via a previous SQLParamData() call.
HY090	Invalid string or buffer length.	The argument <i>DataPtr</i> was not a NULL pointer, and the argument <i>StrLen_or_Ind</i> was less than 0, but not equal to SQL_NTS or SQL_NULL_DATA.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

A additional value for *StrLen\_or\_Ind*, SQL\_DEFAULT\_PARAM, was introduced in ODBC 2.0, to indicate that the procedure is to use the default value of a parameter, rather than a value sent from the application. Since DB2 stored procedure arguments do not support default values, specification of this value for *StrLen\_or\_Ind* argument will result in an error when the CALL statement is executed since the SQL\_DEFAULT\_PARAM value will be considered an invalid length.

ODBC 2.0 also introduced the SQL\_LEN\_DATA\_AT\_EXEC(*length*) macro to be used with the *StrLen\_or\_Ind* argument. The macro is used to specify the sum total length of the entire data that would be sent for character or binary C data via the subsequent SQLPutData() calls. Since the DB2 ODBC driver does not need this information, the macro is not needed. An ODBC application calls SQLGetInfo() with the SQL\_NEED\_LONG\_DATA\_LEN option to check if the driver needs this

## SQLPutData function (CLI) - Passing data value for a parameter

information. The DB2 ODBC driver will return 'N' to indicate that this information is not needed by SQLPutData().

### Example

```
SQLCHAR buffer[BUFSIZ];
size_t n = BUFSIZ;

/* ... */

/* passing data value for a parameter */
cliRC = SQLPutData(hstmt, buffer, n);
```

---

## SQLReloadConfig function (CLI) - Reload a configuration property from the client configuration file

The SQLReloadConfig() function reloads a configuration property from the db2dsdriver.cfg client configuration file.

### Purpose

Specification:	CLI 9.7		
----------------	---------	--	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is SQLReloadConfigW(). For more information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLReloadConfig(SQLINTEGER ConfigProperty,
                           SQLCHAR *DiagInfoString,
                           SQLSMALLINT BufferLength,
                           SQLSMALLINT *StringLengthPtr);
```

### Function arguments

Table 127. SQLReloadConfig function arguments

Data type	Argument	Use	Description
SQLINTEGER	<i>ConfigProperty</i>	Input	A predefined grouping of db2dsdriver.cfg file sections to reload. <b>DSD_ACR_AFFINITY</b> is the only supported value. <b>DSD_ACR_AFFINITY</b> value is defined in the usage section.
SQLCHAR *	<i>DiagInfoString</i>	Output	If the value is SQL_ERROR, CLI returns a detailed error description or information in the <b>DiagInfoString</b> output. If the value is SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, CLI does not return any information.  In DB2 Version 9.7 Fix Pack 5 and later fix packs, warning messages are prefixed with a diagnostic string that consists of the database name, server name, and port number ( <i>database:hostname:port</i> ). Multiple warnings are separated with newline characters.
SQLINTEGER	<i>BufferLength</i>	Input	The length of the <i>*DiagInfoString</i> argument.

## SQLReloadConfig function (CLI) - Reload a configuration property from the client configuration file

Table 127. SQLReloadConfig function arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	<i>StringLengthPtr</i>	Output	A pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the number of bytes that are required for the null termination character, available to return in the <i>*DiagInfoString</i> argument. If the number of bytes to return is greater than the value of the <i>BufferLength</i> argument, the text in the <i>*DiagInfoString</i> argument is truncated to the value of the <i>BufferLength</i> argument minus the length of a null termination character. CLI then uses a null character to terminate the text in the <i>*DiagInfoString</i> argument.

### Usage

After you modify the `db2dsdriver.cfg` file, you can issue the `SQLReloadConfig()` function to reload the entries for the section of the `db2dsdriver.cfg` file that you specify in the *ConfigProperty* argument.

Currently, you can reload only the **DSD\_ACR\_AFFINITY** configuration property. The **DSD\_ACR\_AFFINITY** configuration property consists of the following parameters, which you define in the automatic client reroute (<acr>) section in the `db2dsdriver.cfg` file:

- <alternateserverlist>
- <affinitylist>
- <clientaffinitydefined>
- <clientaffinityroundrobin>

Modifications to other parameters in the <acr> section are ignored.

When your application issues the `SQLReloadConfig()` function, the reload causes reevaluation of affinity members for all connections at the *next transaction interval*. The next transaction interval refers to the next transaction boundary, which is defined when a commit or rollback occurs. This reevaluation of affinity members involves validating entries for all alternate servers within the <acr> section. For each server, an attempt is made to open a socket by using the specified host name and port number. If all servers in the alternate server list of an active database connection are unreachable, an error message is returned in the **DiagInfoString** argument of the `SQLReloadConfig()` function:

```
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - None of the servers,  
specified under <alternateserverlist> section, are reachable.
```

Affinity for an idle connection is evaluated only when the connection becomes active. When the connection becomes active, it is moved to an affinity member.

If you modify a section of the `db2dsdriver.cfg` file other than the <acr> section, the `SQLReloadConfig()` function returns a value of `SQL_ERROR` in the *DiagInfoString* argument. Also, if you remove or add a <acr> section for existing database entries in the `db2dsdriver.cfg` file, the `SQLReloadConfig()` function returns an error. If the `SQLReloadConfig()` function returns an error, applications continue to access the old `db2dsdriver.cfg` file contents.



## SQLReloadConfig function (CLI) - Reload a configuration property from the client configuration file

The SQLReloadConfig() function updates in-memory versions of automatic client reroute (ACR) affinities with all the databases that are listed in the current db2dsdriver.cfg file. If the function finds that the <acr> section of any database entry is invalid, the next attempt to connect to that database results in an error. The detection of invalid <acr> sections also terminates connections for active databases at the next transaction interval. Possible causes for an invalid <acr> section are as follows:

- The <database> section is missing (results in SQL\_ERROR being returned).
- The <acr> section is missing (results in SQL\_ERROR being returned).
- A new alternate server list is empty.
- CLI encountered internal buffer boundary limit to hold new server list.

The SQLReloadConfig() function has no default timeout value for attempting to open a socket against an alternate server port. You can set a specific timeout value by using the **SocketTimeOut** parameter in the global section of the db2dsdriver.cfg file.

### Return codes

- SQL\_ERROR
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

The following tables provide lists of errors and warnings that can result from calling the SQLReloadConfig() function.

Table 128. SQLReloadConfig() function error messages

Error message	Explanation
IBM DB2 [CLI Driver] Another SQLReloadConfig() execution is already under progress.	The SQLReloadConfig() function was already called for a particular process and is running.
IBM DB2 [CLI Driver] Unrecognized value found in ConfigProperty argument of SQLReloadConfig().	A value other than the <b>DSD_ACR_AFFINITY</b> configuration property was specified.
IBM DB2 [CLI Driver] db2dsdriver.cfg file to be reloaded cannot be located in the expected location.	The db2dsdriver.cfg file cannot be accessed. Ensure that the db2dsdriver.cfg file is present and has global read permission.
IBM DB2 [CLI Driver] Sections other than one specified in ConfigProperty argument is found modified, which is not supported.	A section other than the <acr> section was modified.
IBM DB2 [CLI Driver] CLI subsystem is not initialized. Use SQLAllocHandle() to allocate environment handle.	You must allocate the environment handle before calling the SQLReloadConfig() function.
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - Either all or one of <client>, <affinitylist>, <alternateserverlist> sections are missing in db2dsdriver.cfg.	The db2dsdriver.cfg file is missing an entry for all or one of the <client>, <affinitylist>, and <alternateserverlist> sections.

## SQLReloadConfig function (CLI) - Reload a configuration property from the client configuration file

Table 128. SQLReloadConfig() function error messages (continued)

Error message	Explanation
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - None of the servers, specified under <alternateserverlist> section, are reachable.	The IBM driver cannot establish connection to all the servers that are specified in the <alternateserverlist> section of the db2dsdriver.cfg file.
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - cannot find appropriate port number for service name <srvcname>.	The IBM driver cannot allocate a port number for the service name that is specified in the error message.

Table 129. SQLReloadConfig() function warning message

Error message	Explanation
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - The following server(s) from <alternateserverlist> were unreachable "<hostname>:<port>,<hostname>:port,...".	The IBM driver cannot establish connection to some of the servers that were specified in the <alternateserverlist> section of the db2dsdriver.cfg file.

### Restrictions

The **DSD\_ACR\_AFFINITY** configuration property of the db2dsdriver.cfg file is the only property that you can reload by using the SQLReloadConfig() function.

---

## SQLRowCount function (CLI) - Get row count

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI

SQLRowCount() returns the number of rows in a table that were affected by an UPDATE, an INSERT, a DELETE, or a MERGE statement issued against the table, or a view based on the table.

You must call SQLExecute() or SQLExecDirect() before calling SQLRowCount().

### Syntax

```
SQLRETURN SQLRowCount (
                SQLHSTMT StatementHandle, /* hstmt */
                SQLLEN *RowCountPtr); /* pcrow */
```

### Function arguments

Table 130. SQLRowCount arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle
SQLLEN *	RowCountPtr	output	Pointer to location where the number of rows affected is stored. If the SQL_ATTR_PARC_BATCH connection attribute is set to SQL_PARC_BATCH_ENABLE, the location size must be the size of an array. See the Usage section for details.

## Usage

If the last issued statement referenced by the input statement handle was not an UPDATE, an INSERT, a DELETE, or a MERGE statement or if the statement did not run successfully, the function sets the contents of *StatementHandle* to -1.

If you use the SQLRowCount() function on a non-scrollable SELECT-only cursor, the function sets the contents of *RowCountPtr* to -1. The number of rows is not available until all of the data has been fetched. You can use the CLI statement attribute SQL\_ATTR\_ROWCOUNT\_PREFETCH to enable the client to request the full row count before fetching the data.

**Restriction:** The SQL\_ATTR\_ROWCOUNT\_PREFETCH attribute is not supported when the cursor contains LOBs or XML.

If the SQL\_ATTR\_PARC\_BATCH connection attribute is set to SQL\_PARC\_BATCH\_ENABLE, then the SQL\_ATTR\_PARAMOPT\_ATOMIC attribute must be set to SQL\_ATOMIC\_NO and the *RowCountPtr* argument must be pointing to an array of type SQLLEN \*. The length of this array must be equal to SQL\_ATTR\_PARAMSET\_SIZE. Upon successful execution of non-atomic update, delete or insert operations, the number of rows in a table that were affected by each parameter set is stored in this array.

Any rows in other tables that might have been affected by the statement (for example, due to cascading deletes) are not included in the count.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 131. SQLRowCount SQLSTATEs

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called before calling SQLExecute() or SQLExecDirect() for the <i>StatementHandle</i> .
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

## Authorization

None.

## SQLSetColAttributes function (CLI) - Set column attributes

### Deprecated

#### Note:

In ODBC 3.0, SQLSetColAttributes() has been deprecated, and CLI no longer supports this function.

Now that CLI uses deferred prepare by default, there is no need for the functionality of SQLSetColAttributes().

## SQLSetConnectAttr function (CLI) - Set connection attributes

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLSetConnectAttr() sets attributes that govern aspects of connections.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSetConnectAttrW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLSetConnectAttr (
    SQLHDBC          ConnectionHandle, /* hdbc */
    SQLINTEGER       Attribute,        /* fOption */
    SQLPOINTER       ValuePtr,        /* pvParam */
    SQLINTEGER       StringLength);   /* fStrLen */
```

### Function arguments

Table 132. SQLSetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle.
SQLINTEGER	<i>Attribute</i>	input	Attribute to set, listed in the connection attributes list.
SQLPOINTER	<i>ValuePtr</i>	input	Pointer to the value to be associated with <i>Attribute</i> . Depending on the value of <i>Attribute</i> , <i>ValuePtr</i> will be a 32-bit unsigned integer value or pointer to a null-terminated character string. Note that if the <i>Attribute</i> argument is a driver-specific value, the value in <i>*ValuePtr</i> can be a signed integer. Refer to the connection attributes list for details.

## SQLSetConnectAttr function (CLI) - Set connection attributes

Table 132. SQLSetConnectAttr arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>StringLength</i>	input	<p>If <i>Attribute</i> is an ODBC-defined attribute and <i>ValuePtr</i> points to a character string or a binary buffer, this argument should be the length of <i>*ValuePtr</i>. For character string data, <i>StringLength</i> should contain the number of bytes in the string. If <i>Attribute</i> is an ODBC-defined attribute and <i>ValuePtr</i> is an integer, <i>StringLength</i> is ignored.</p> <p>If <i>Attribute</i> is a CLI attribute, the application indicates the nature of the attribute by setting the <i>StringLength</i> argument. <i>StringLength</i> can have the following values:</p> <ul style="list-style-type: none"><li>• If <i>ValuePtr</i> is a pointer to a character string, then <i>StringLength</i> is the number of bytes needed to store the string or SQL_NTS.</li><li>• If <i>ValuePtr</i> is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in <i>StringLength</i>. This places a negative value in <i>StringLength</i>.</li><li>• If <i>ValuePtr</i> is a pointer to a value other than a character string or a binary string, then <i>StringLength</i> should have the value SQL_IS_POINTER.</li><li>• If <i>ValuePtr</i> contains a fixed-length value, then <i>StringLength</i> is either SQL_IS_INTEGER or SQL_IS_UIINTEGER, as appropriate.</li></ul>

### Usage

#### Setting statement attributes using SQLSetConnectAttr() no longer supported

The ability to set statement attributes using SQLSetConnectAttr() is no longer supported. To support applications written before version 5, some statement attributes can be set using SQLSetConnectAttr() in this release of CLI. All applications that rely on this behavior, however, should be updated to use SQLSetStmtAttr() instead.

If SQLSetConnectAttr() is called to set a statement attribute that sets the header field of a descriptor, the descriptor field is set for the application descriptors currently associated with all statements on the connection. However, the attribute setting does not affect any descriptors that might be associated with the statements on that connection in the future.

#### Connection Attributes

At any time between allocating and freeing a connection, an application can call SQLSetConnectAttr(). All connection and statement attributes successfully set by the application for the connection persist until SQLFreeHandle() is called on the connection.

Some connection attributes can be set only before a connection has been made; others can be set only after a connection has been made, while some cannot be set once a statement is allocated. Refer to the connection attributes list for details on when each attribute can be set.

## SQLSetConnectAttr function (CLI) - Set connection attributes

Some connection attributes support substitution of a similar value if the data source does not support the value specified in *ValuePtr*. In such cases, CLI returns SQL\_SUCCESS\_WITH\_INFO and SQLSTATE 01S02 (Option value changed.). To determine the substituted value, an application calls SQLGetConnectAttr().

The format of information set through *ValuePtr* depends on the specified *Attribute*. SQLSetConnectAttr() will accept attribute information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of each is noted in the attribute's description. Character strings pointed to by the *ValuePtr* argument of SQLSetConnectAttr() have a length of *StringLength* bytes. The *StringLength* argument is ignored if the length is defined by the attribute.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

CLI can return SQL\_SUCCESS\_WITH\_INFO to provide information about the result of setting an option.

When *Attribute* is a statement attribute, SQLSetConnectAttr() can return any SQLSTATEs returned by SQLSetStmtAttr().

Table 133. SQLSetConnectAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	General error.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed.	CLI did not support the value specified in <i>*ValuePtr</i> and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08002	Connection in use.	The argument <i>Attribute</i> was SQL_ATTR_ODBC_CURSORS and CLI was already connected to the data source.
08003	Connection is closed.	An <i>Attribute</i> value was specified that required an open connection, but the <i>ConnectionHandle</i> was not in a connected state.
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
24000	Invalid cursor state.	The argument <i>Attribute</i> was SQL_ATTR_CURRENT_QUALIFIER and a result set was pending.
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by SQLGetDiagRec() in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	A null pointer was passed for <i>ValuePtr</i> and the value in <i>*ValuePtr</i> was a string value.

## SQLSetConnectAttr function (CLI) - Set connection attributes

Table 133. SQLSetConnectAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>An asynchronously executing function was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and was still executing when <code>SQLSetConnectAttr()</code> was called.</p> <p><code>SQLExecute()</code> or <code>SQLExecDirect()</code> was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and returned <code>SQL_NEED_DATA</code>. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p><code>SQLBrowseConnect()</code> was called for the <i>ConnectionHandle</i> and returned <code>SQL_NEED_DATA</code>. This function was called before <code>SQLBrowseConnect()</code> returned <code>SQL_SUCCESS_WITH_INFO</code> or <code>SQL_SUCCESS</code>.</p>
HY011	Operation invalid at this time.	The argument <i>Attribute</i> was <code>SQL_ATTR_TXN_ISOLATION</code> and a transaction was open.
HY024	Invalid attribute value.	<p>Given the specified <i>Attribute</i> value, an invalid value was specified in <i>*ValuePtr</i>. (CLI returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as <code>SQL_ATTR_ACCESS_MODE</code>. For all other connection and statement attributes, CLI must verify the value specified in <i>ValuePtr</i>.)</p> <p>The <i>Attribute</i> argument was <code>SQL_ATTR_TRACEFILE</code> or <code>SQL_ATTR_TRANSLATE_LIB</code>, and <i>*ValuePtr</i> was an empty string.</p>
HY090	Invalid string or buffer length.	The <i>StringLength</i> argument was less than 0, but was not <code>SQL_NTS</code> .
HY092	Option type out of range.	The value specified for the argument <i>Attribute</i> was not valid for this version of CLI.
HYC00	Driver not capable.	The value specified for the argument <i>Attribute</i> was a valid connection or statement attribute for the version of the CLI driver, but was not supported by the data source.

### Restrictions

None.

### Example

```
/* set AUTOCOMMIT on */
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_AUTOCOMMIT,
                          (SQLPOINTER)SQL_AUTOCOMMIT_ON,
                          SQL_NTS);

/* ... */

/* set AUTOCOMMIT OFF */
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_AUTOCOMMIT,
                          (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
                          SQL_NTS);
```

## SQLSetConnection function (CLI) - Set connection handle

### Purpose

Specification:	CLI 2.1		
----------------	---------	--	--

## SQLSetConnection function (CLI) - Set connection handle

This function is needed if the application needs to deterministically switch to a particular connection before continuing execution. It should only be used when the application is mixing CLI function calls with embedded SQL function calls and where multiple connections are used.

### Syntax

```
SQLRETURN SQLSetConnection (SQLHDBC          ConnectionHandle); /* hdbc */
```

### Function arguments

Table 134. SQLSetConnection arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	The connection handle associated with the connection that the application wishes to switch to.

### Usage

In CLI version 1 it was possible to mix CLI calls with calls to routines containing embedded SQL as long as the connect request was issued via the CLI connect function. The embedded SQL routine would simply use the existing CLI connection.

Although this is still true, there is a potential complication: CLI allows multiple concurrent connections. This means that it is no longer clear which connection an embedded SQL routine would use upon being invoked. In practice, the embedded routine would use the connection associated with the most recent network activity. However, from the application's perspective, this is not always deterministic and it is difficult to keep track of this information. SQLSetConnection() is used to allow the application to *explicitly* specify which connection is active. The application can then call the embedded SQL routine.

SQLSetConnection() is not needed if the application makes use of CLI exclusively. Under those conditions, each statement handle is implicitly associated with a connection handle and there is never any confusion as to which connection a particular CLI function applies.

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 135. SQLSetConnection SQLSTATEs

SQLSTATE	Description	Explanation
08003	Connection is closed.	The connection handle provided is not currently associated with an open connection to a database server.
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by SQLGetDiagRec() in the argument <i>MessageText</i> describes the error and its cause.



## SQLSetConnection function (CLI) - Set connection handle

### Restrictions

None.

### Example

```
/* perform statements on the first connection */
cliRC = SQLSetConnection(hdbc1);

/* ... */

/* perform statements on the second connection */
cliRC = SQLSetConnection(hdbc2);
```

---

## SQLSetConnectOption function (CLI) - Set connection option

### Deprecated

#### Note:

In ODBC 3.0, `SQLSetConnectOption()` has been deprecated and replaced with `SQLSetConnectAttr()`.

Although this version of CLI continues to support `SQLSetConnectOption()`, use `SQLSetConnectAttr()` in your CLI programs so that they conform to the latest standards.

This deprecated function cannot be used in a 64-bit environment.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLSetConnectOptionW()`. Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Migrating to the new function

The statement:

```
SQLSetConnectOption(
    hdbc,
    SQL_AUTOCOMMIT,
    SQL_AUTOCOMMIT_OFF);
```

for example, would be rewritten using the new function as:

```
SQLSetConnectAttr(
    hdbc,
    SQL_ATTR_AUTOCOMMIT,
    SQL_AUTOCOMMIT_OFF,
    0);
```

---

## SQLSetCursorName function (CLI) - Set cursor name

### Purpose

Specification:	CLI 1.1	ODBC 1.0	ISO CLI
----------------	---------	----------	---------

## SQLSetCursorName function (CLI) - Set cursor name

SQLSetCursorName() associates a cursor name with the statement handle. This function is optional because CLI implicitly generates a cursor name. The implicit cursor name is available after the dynamic SQL has been prepared on the statement handle.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSetCursorNameW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLSetCursorName (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *CursorName, /* szCursor */
    SQLSMALLINT NameLength); /* cbCursor */
```

### Function arguments

Table 136. SQLSetCursorName arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle
SQLCHAR *	<i>CursorName</i>	input	Cursor name
SQLSMALLINT	<i>NameLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the <i>CursorName</i> argument.

### Usage

CLI always generates and uses an internally generated cursor name when a query is prepared or executed directly. SQLSetCursorName() allows an application-defined cursor name to be used in an SQL statement (a positioned UPDATE or DELETE). CLI maps this name to the internal name. The name will remain associated with the statement handle, until the handle is dropped, or another SQLSetCursorName() is called on this statement handle.

Although SQLGetCursorName() will return the name set by the application (if one was set), error messages associated with positioned UPDATE and DELETE statements will refer to the internal name. For this reason, do not use SQLSetCursorName() for positioned UPDATES and DELETES, but instead use the internal name which can be obtained by calling SQLGetCursorName().

Cursor names must follow these rules:

- All cursor names within the connection must be unique.
- Each cursor name must be less than or equal to 128 bytes in length. Any attempt to set a cursor name longer than 128 bytes results in truncation of that cursor name to 128 bytes. (No warning is generated.)
- Since internally generated names begin with SQLCUR or SQL\_CUR, the application must not input a cursor name starting with either SQLCUR or SQL\_CUR in order to avoid conflicts with internal names.
- Since a cursor name is considered an identifier in SQL, it must begin with an English letter (a-z, A-Z) followed by any combination of digits (0-9), English letters or the underscore character (\_).

## SQLSetCursorName function (CLI) - Set cursor name

- To permit cursor names containing characters other than those listed above (such as National Language Set or Double Bytes Character Set characters), the application must enclose the cursor name in double quotes ("").
- Unless the input cursor name is enclosed in double quotes, all leading and trailing blanks from the input cursor name string will be removed.

For efficient processing, applications should not include any leading or trailing spaces in the *CursorName* buffer. If the *CursorName* buffer contains a delimited identifier, applications should position the first double quote as the first character in the *CursorName* buffer.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 137. SQLSetCursorName SQLSTATES

SQLSTATE	Description	Explanation
34000	Invalid cursor name.	<p>The cursor name specified by the argument <i>CursorName</i> was invalid. The cursor name either begins with "SQLCUR" or "SQL_CUR" or violates the cursor naming rules (Must begin with a-z or A-Z followed by any combination of English letters, digits, or the '_' character.</p> <p>The cursor name specified by the argument <i>CursorName</i> already exists.</p> <p>The cursor name length is greater than the value returned by SQLGetInfo() with the SQL_MAX_CURSOR_NAME_LEN argument.</p>
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	<i>CursorName</i> was a null pointer.
HY010	Function sequence error.	<p>There is an open or positioned cursor on the statement handle.</p> <p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>

## SQLSetCursorName function (CLI) - Set cursor name

Table 137. SQLSetCursorName SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The argument <i>NameLength</i> was less than 0, but not equal to SQL_NTS.

### Authorization

None.

### Example

```
/* set the name of the cursor */  
rc = SQLSetCursorName(hstmtSelect, (SQLCHAR *)"CURSNAME", SQL_NTS);
```

## SQLSetDescField function (CLI) - Set a single field of a descriptor record

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLSetDescField() sets the value of a single field of a descriptor record.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSetDescFieldW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLSetDescField (SQLHDESC      DescriptorHandle,  
                           SQLSMALLINT   RecNumber,  
                           SQLSMALLINT   FieldIdentifier,  
                           SQLPOINTER    ValuePtr,  
                           SQLINTEGER    BufferLength);
```

### Function arguments

Table 138. SQLSetDescField arguments

Data type	Argument	Use	Description
SQLHDESC	<i>DescriptorHandle</i>	input	Descriptor handle.
SQLSMALLINT	<i>RecNumber</i>	input	Indicates the descriptor record containing the field that the application seeks to set. Descriptor records are numbered from 0, with record number 0 being the bookmark record. The <i>RecNumber</i> argument is ignored for header fields.
SQLSMALLINT	<i>FieldIdentifier</i>	input	Indicates the field of the descriptor whose value is to be set. For more information, refer to the list of values for the descriptor <i>FieldIdentifier</i> argument.

## SQLSetDescField function (CLI) - Set a single field of a descriptor record

Table 138. *SQLSetDescField* arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	<i>ValuePtr</i>	input	Pointer to a buffer containing the descriptor information, or a four-byte value. The data type depends on the value of <i>FieldIdentifier</i> . If <i>ValuePtr</i> is a four-byte value, either all four of the bytes are used, or just two of the four are used, depending on the value of the <i>FieldIdentifier</i> argument.
SQLINTEGER	<i>BufferLength</i>	input	<p>If <i>FieldIdentifier</i> is an ODBC-defined field and <i>ValuePtr</i> points to a character string or a binary buffer, this argument should be the length of <i>*ValuePtr</i>. For character string data, <i>BufferLength</i> should contain the number of bytes in the string. If <i>FieldIdentifier</i> is an ODBC-defined field and <i>ValuePtr</i> is an integer, <i>BufferLength</i> is ignored.</p> <p>If <i>FieldIdentifier</i> is a driver-defined field, the application indicates the nature of the field by setting the <i>BufferLength</i> argument. <i>BufferLength</i> can have the following values:</p> <ul style="list-style-type: none"> <li>• If <i>ValuePtr</i> is a pointer to a character string, then <i>BufferLength</i> is the number of bytes needed to store the string or SQL_NTS.</li> <li>• If <i>ValuePtr</i> is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in <i>BufferLength</i>. This places a negative value in <i>BufferLength</i>.</li> <li>• If <i>ValuePtr</i> is a pointer to a value other than a character string or a binary string, then <i>BufferLength</i> should have the value SQL_IS_POINTER.</li> <li>• If <i>ValuePtr</i> contains a fixed-length value, then <i>BufferLength</i> is either SQL_IS_INTEGER, SQL_IS_UIINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.</li> </ul>

### Usage

An application can call `SQLSetDescField()` to set any descriptor field one at a time. One call to `SQLSetDescField()` sets a single field in a single descriptor. This function can be called to set any field in any descriptor type, provided the field can be set. See the descriptor header and record field initialization values for more information.

**Note:** If a call to `SQLSetDescField()` fails, the contents of the descriptor record identified by the *RecNumber* argument are undefined.

Other functions can be called to set multiple descriptor fields with a single call of the function. The `SQLSetDescRec()` function sets a variety of fields that affect the data type and buffer bound to a column or parameter (the `SQL_DESC_TYPE`, `SQL_DESC_DATETIME_INTERVAL_CODE`, `SQL_DESC_OCTET_LENGTH`, `SQL_DESC_PRECISION`, `SQL_DESC_SCALE`, `SQL_DESC_DATA_PTR`, `SQL_DESC_OCTET_LENGTH_PTR`, and `SQL_DESC_INDICATOR_PTR` fields). `SQLBindCol()` or `SQLBindParameter()` can be used to make a complete specification

## SQLSetDescField function (CLI) - Set a single field of a descriptor record

for the binding of a column or parameter. These functions each set a specific group of descriptor fields with one function call.

SQLSetDescField() can be called to change the binding buffers by adding an offset to the binding pointers (SQL\_DESC\_DATA\_PTR, SQL\_DESC\_INDICATOR\_PTR, or SQL\_DESC\_OCTET\_LENGTH\_PTR). This changes the binding buffers without calling SQLBindCol() or SQLBindParameter(). This allows an application to quickly change SQL\_DESC\_DATA\_PTR without concern for changing other fields, for instance SQL\_DESC\_DATA\_TYPE.

Descriptor header fields are set by calling SQLSetDescField() with a *RecNumber* of 0, and the appropriate *FieldIdentifier*. Many header fields contain statement attributes, so can also be set by a call to SQLSetStmtAttr(). This allows applications to set a statement attribute without first obtaining a descriptor handle. A *RecNumber* of 0 is also used to set bookmark fields.

**Note:** The statement attribute SQL\_ATTR\_USE\_BOOKMARKS should always be set before calling SQLSetDescField() to set bookmark fields. While this is not mandatory, it is strongly recommended.

### Sequence of setting descriptor fields

When setting descriptor fields by calling SQLSetDescField(), the application must follow a specific sequence:

- The application must first set the SQL\_DESC\_TYPE, SQL\_DESC\_CONCISE\_TYPE, or SQL\_DESC\_DATETIME\_INTERVAL\_CODE field.

**Note:** SQL\_DESC\_DATETIME\_INTERVAL\_CODE is defined by ODBC but not supported by CLI.

- After one of these fields has been set, the application can set an attribute of a data type, and the driver sets data type attribute fields to the appropriate default values for the data type. Automatic defaulting of type attribute fields ensures that the descriptor is always ready to use once the application has specified a data type. If the application explicitly sets a data type attribute, it is overriding the default attribute.
- After one of the fields listed in Step 1 has been set, and data type attributes have been set, the application can set SQL\_DESC\_DATA\_PTR. This prompts a consistency check of descriptor fields. If the application changes the data type or attributes after setting the SQL\_DESC\_DATA\_PTR field, then the driver sets SQL\_DESC\_DATA\_PTR to a null pointer, unbinding the record. This forces the application to complete the proper steps in sequence, before the descriptor record is usable.

### Initialization of descriptor fields

When a descriptor is allocated, the fields in the descriptor can be initialized to a default value, be initialized without a default value, or be undefined for the type of descriptor. Refer to the list of descriptor header and record field initialization values for details.

The fields of an IRD have a default value only after the statement has been prepared or executed and the IRD has been populated, not when the statement handle or descriptor has been allocated. Until the IRD has been populated, any attempt to gain access to a field of an IRD will return an error.

## SQLSetDescField function (CLI) - Set a single field of a descriptor record

Some descriptor fields are defined for one or more, but not all, of the descriptor types (ARDs and IRDs, and APDs and IPDs). When a field is undefined for a type of descriptor, it is not needed by any of the functions that use that descriptor. Because a descriptor is a logical view of data, rather than an actual data structure, these extra fields have no effect on the defined fields.

The fields that can be accessed by `SQLGetDescField()` are not necessarily set by `SQLSetDescField()`. Fields that can be set by `SQLSetDescField()` are described in the descriptor header and record field initialization values list.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 139. *SQLSetDescField* SQLSTATES

SQLSTATE	Description	Explanation
01000	General warning	Informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01S02	Option value changed.	CLI did not support the value specified in <i>*ValuePtr</i> (if <i>ValuePtr</i> was a pointer) or the value in <i>ValuePtr</i> (if <i>ValuePtr</i> was a four-byte value), or <i>*ValuePtr</i> was invalid because of SQL constraints or requirements, so CLI substituted a similar value. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
07009	Invalid descriptor index.	The <i>FieldIdentifier</i> argument was a header field, and the <i>RecNumber</i> argument was not 0.  The <i>RecNumber</i> argument was 0 and the <i>DescriptorHandle</i> was an IPD.  The <i>RecNumber</i> argument was less than 0.
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by <code>SQLGetDiagRec()</code> in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called.  <code>SQLExecute()</code> or <code>SQLExecDirect()</code> was called for the <i>StatementHandle</i> with which the <i>DescriptorHandle</i> was associated and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
HY016	Cannot modify an implementation row descriptor.	The <i>DescriptorHandle</i> argument was associated with an IRD, and the <i>FieldIdentifier</i> argument was not <code>SQL_DESC_ARRAY_STATUS_PTR</code> .

## SQLSetDescField function (CLI) - Set a single field of a descriptor record

Table 139. SQLSetDescField SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY021	Inconsistent descriptor information.	The TYPE field, or any other field associated with the TYPE field in the descriptor, was not valid or consistent. The TYPE field was not a valid CLI C type.  Descriptor information checked during a consistency check was not consistent.
HY091	Invalid descriptor field identifier.	The value specified for the <i>FieldIdentifier</i> argument was not a CLI defined field and was not a defined value.  The value specified for the <i>RecNumber</i> argument was greater than the value in the SQL_DESC_COUNT field.  The <i>FieldIdentifier</i> argument was SQL_DESC_ALLOC_TYPE.
HY092	Option type out of range.	The value specified for the <i>Attribute</i> argument was not valid.
HY094	Invalid scale value.	The value specified for <i>pfParamType</i> was either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>DecimalDigits</i> was less than 0 or greater than the value for the argument <i>pcbColDef</i> (precision).  The value specified for <i>pfParamType</i> was SQL_C_TYPE_TIMESTAMP and the value for <i>pfParamType</i> was either SQL_CHAR or SQL_VARCHAR and the value for <i>DecimalDigits</i> was less than 0 or greater than 9.  The value specified for <i>pfParamType</i> was SQL_C_TIMESTAMP_EXT and the value for <i>DecimalDigits</i> was less than 0 or greater than 12.
HY105	Invalid parameter type.	The value specified for the SQL_DESC_PARAMETER_TYPE field was invalid. (For more information, see the <i>InputOutputType</i> Argument section in SQLBindParameter().)

### Restrictions

None.

### Example

```
/* set a single field of a descriptor record */
rc = SQLSetDescField(hARD,
                    1,
                    SQL_DESC_TYPE,
                    (SQLPOINTER)SQL_SMALLINT,
                    SQL_IS_SMALLINT);
```

## SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI

The SQLSetDescRec() function sets multiple descriptor fields that affect the data type and buffer bound to a column or parameter data.



## SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data

### Syntax

```
SQLRETURN SQLSetDescRec (SQLHDESC
                        SQLSMALLINT
                        SQLSMALLINT
                        SQLSMALLINT
                        SQLLEN
                        SQLSMALLINT
                        SQLSMALLINT
                        SQLPOINTER
                        SQLLEN
                        SQLLEN
                        DescriptorHandle,
                        RecNumber,
                        Type,
                        SubType,
                        Length,
                        Precision,
                        Scale,
                        DataPtr,
                        *StringLengthPtr,
                        *IndicatorPtr);
```

### Function arguments

Table 140. SQLSetDescRec arguments

Data type	Argument	Use	Description
SQLHDESC	<i>DescriptorHandle</i>	input	Descriptor handle. This must not be an IRD handle.
SQLSMALLINT	<i>RecNumber</i>	input	Indicates the descriptor record that contains the fields to be set. Descriptor records are numbered from 0, with record number 0 being the bookmark record. This argument must be equal to or greater than 0. If <i>RecNumber</i> is greater than the value of SQL_DESC_COUNT, SQL_DESC_COUNT is changed to the value of <i>RecNumber</i> .
SQLSMALLINT	<i>Type</i>	input	The value to which to set the SQL_DESC_TYPE field for the descriptor record.
SQLSMALLINT	<i>SubType</i>	input	For records whose type is SQL_DATETIME, this is the value to which to set the SQL_DESC_DATETIME_INTERVAL_CODE field.
SQLLEN	<i>Length</i>	input	The value to which to set the SQL_DESC_OCTET_LENGTH field for the descriptor record.
SQLSMALLINT	<i>Precision</i>	input	The value to which to set the SQL_DESC_PRECISION field for the descriptor record.
SQLSMALLINT	<i>Scale</i>	input	The value to which to set the SQL_DESC_SCALE field for the descriptor record.
SQLPOINTER	<i>DataPtr</i>	Deferred Input or Output	The value to which to set the SQL_DESC_DATA_PTR field for the descriptor record. <i>DataPtr</i> can be set to a null pointer to set the SQL_DESC_DATA_PTR field to a null pointer.
SQLLEN *	<i>StringLengthPtr</i>	Deferred Input or Output	The value to which to set the SQL_DESC_OCTET_LENGTH_PTR field for the descriptor record. <i>StringLengthPtr</i> can be set to a null pointer to set the SQL_DESC_OCTET_LENGTH_PTR field to a null pointer.
SQLLEN *	<i>IndicatorPtr</i>	Deferred Input or Output	The value to which to set the SQL_DESC_INDICATOR_PTR field for the descriptor record. <i>IndicatorPtr</i> can be set to a null pointer to set the SQL_DESC_INDICATOR_PTR field to a null pointer.

## SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data

### Usage

An application can call `SQLSetDescRec()` to set the following fields for a single column or parameter:

- `SQL_DESC_TYPE`
- `SQL_DESC_OCTET_LENGTH`
- `SQL_DESC_PRECISION`
- `SQL_DESC_SCALE`
- `SQL_DESC_DATA_PTR`
- `SQL_DESC_OCTET_LENGTH_PTR`
- `SQL_DESC_INDICATOR_PTR`

`SQL_DESC_DATETIME_INTERVAL_CODE` can only be updated if `SQL_DESC_TYPE` indicates `SQL_DATETIME`.

**Note:** If a call to `SQLSetDescRec()` fails, the contents of the descriptor record identified by the *RecNumber* argument are undefined.

When binding a column or parameter, `SQLSetDescRec()` allows you to change multiple fields affecting the binding without calling `SQLBindCol()` or `SQLBindParameter()`, or making multiple calls to `SQLSetDescField()`. `SQLSetDescRec()` can set fields on a descriptor not currently associated with a statement. Note that `SQLBindParameter()` sets more fields than `SQLSetDescRec()`, can set fields on both an APD and an IPD in one call, and does not require a descriptor handle.

The statement attribute `SQL_ATTR_USE_BOOKMARKS` should always be set before calling `SQLSetDescRec()` with a *RecNumber* argument of 0 to set bookmark fields. While this is not mandatory, it is strongly recommended.

### Return Codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

### Diagnostics

Table 141. `SQLSetDescRec` `SQLSTATES`

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
07009	Invalid descriptor index.	The <i>RecNumber</i> argument was set to 0, and the <i>DescriptorHandle</i> was an IPD handle.  The <i>RecNumber</i> argument was less than 0.  The <i>RecNumber</i> argument was greater than the maximum number of columns or parameters that the data source can support, and the <i>DescriptorHandle</i> argument was an APD, IPD, or ARD.  The <i>RecNumber</i> argument was equal to 0, and the <i>DescriptorHandle</i> argument referred to an implicitly allocated APD. (This error does not occur with an explicitly allocated application descriptor, because it is not known whether an explicitly allocated application descriptor is an APD or ARD until execute time.)

## SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data

Table 141. SQLSetDescRec SQLSTATEs (continued)

SQLSTATE	Description	Explanation
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	<p>The <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called.</p> <p>SQLExecute() or SQLExecDirect() was called for the <i>StatementHandle</i> with which the <i>DescriptorHandle</i> was associated and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters.</p>
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY016	Cannot modify an implementation row descriptor.	The <i>DescriptorHandle</i> argument was associated with an IRD.
HY021	Inconsistent descriptor information.	<p>The <i>Type</i> field, or any other field associated with the <i>TYPE</i> field in the descriptor, was not valid or consistent.</p> <p>Descriptor information checked during a consistency check was not consistent.</p>
HY094	Invalid scale value.	<p>The value specified for <i>pfParamType</i> was either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>DecimalDigits</i> was less than 0 or greater than the value for the argument <i>pcbColDef</i> (precision).</p> <p>The value specified for <i>pfParamType</i> was SQL_C_TYPE_TIMESTAMP and the value for <i>pfParamType</i> was either SQL_CHAR or SQL_VARCHAR and the value for <i>DecimalDigits</i> was less than 0 or greater than 9.</p> <p>The value specified for <i>pfParamType</i> was SQL_C_TIMESTAMP_EXT and the value for <i>DecimalDigits</i> was less than 0 or greater than 12.</p>

### Restrictions

None.

### Example

```
SQLSMALLINT type;  
SQLINTEGER length, datalen;  
SQLSMALLINT id_no;  
/* ... */  
  
/* set multiple descriptor fields for a column or parameter data */  
rc = SQLSetDescRec(hARD, 1, type, 0, length, 0, 0, &id_no, &datalen, NULL);
```

## SQLSetEnvAttr function (CLI) - Set environment attribute

### Purpose

Specification:	CLI 2.1		ISO CLI
----------------	---------	--	---------

SQLSetEnvAttr() sets an environment attribute for the current environment.

### Syntax

```
SQLRETURN SQLSetEnvAttr (SQLHENV EnvironmentHandle, /* henv */
                        SQLINTEGER Attribute,
                        SQLPOINTER ValuePtr, /* Value */
                        SQLINTEGER StringLength);
```

### Function arguments

Table 142. SQLSetEnvAttr arguments

Data type	Argument	Use	Description
SQLHENV	<i>EnvironmentHandle</i>	Input	Environment handle.
SQLINTEGER	<i>Attribute</i>	Input	Environment attribute to set; refer to the list of CLI environment attributes for descriptions.
SQLPOINTER	<i>ValuePtr</i>	Input	The desired value for <i>Attribute</i> .
SQLINTEGER	<i>StringLength</i>	Input	Length of <i>ValuePtr</i> in bytes if the attribute value is a character string; if <i>Attribute</i> does not denote a string, then CLI ignores <i>StringLength</i> .

### Usage

Once set, the attribute's value affects all connections under this environment.

The application can obtain the current attribute value by calling SQLGetEnvAttr().

Refer to the list of CLI environment attributes for the attributes that can be set with SQLSetEnvAttr().

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 143. SQLSetEnvAttr SQLSTATES

SQLSTATE	Description	Explanation
HY011	Operation invalid at this time.	Applications cannot set environment attributes while connection handles are allocated on the environment handle.
HY024	Invalid attribute value	Given the specified <i>Attribute</i> value, an invalid value was specified in <i>*ValuePtr</i> .
HY090	Invalid string or buffer length	The <i>StringLength</i> argument was less than 0, but was not SQL_NTS.

## SQLSetEnvAttr function (CLI) - Set environment attribute

Table 143. SQLSetEnvAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Option type out of range.	An invalid <i>Attribute</i> value was specified.
HYC00	Driver not capable.	The specified <i>Attribute</i> is not supported by CLI.  Given specified <i>Attribute</i> value, the value specified for the argument <i>ValuePtr</i> is not supported.

### Restrictions

None.

### Example

```
/* set environment attribute */  
cliRC = SQLSetEnvAttr(henv, SQL_ATTR_OUTPUT_NTS, (SQLPOINTER) SQL_TRUE, 0);
```

## SQLSetParam function (CLI) - Bind a parameter marker to a buffer or LOB locator

### Deprecated

#### Note:

In ODBC 2.0 and above, SQLSetParam() is deprecated and replaced with SQLBindParameter().

Although this version of CLI continues to support SQLSetParam(), use SQLBindParameter() in your CLI programs so that they conform to the latest standards.

#### Equivalent function: SQLBindParameter()

The CLI function SQLBindParameter() is functionally the same as the SQLSetParam() function. Both take a similar number and type of arguments, behave the same, and return the same return codes. The difference is that SQLSetParam() does not have the *InputOutputType* or *BufferLength* arguments to specify the parameter type and maximum buffer length. Calling SQLSetParam() is functionally equivalent to calling SQLBindParameter() with the *InputOutputType* argument set to SQL\_PARAM\_INPUT and the *BufferLength* argument set to SQL\_SETPARAM\_VALUE\_MAX.

#### Migrating to the new function

The statement:

```
SQLSetParam(hstmt, 1, SQL_C_SHORT, SQL_SMALLINT, 0, 0,  
            &parameter1, NULL);
```

for example, would be rewritten using the new function as:

```
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,  
                SQL_SMALLINT, 0, 0, &parameter1,  
                SQL_SETPARAM_VALUE_MAX, NULL);
```

## SQLSetPos function (CLI) - Set the cursor position in a rowset

### Purpose

Specification:	CLI 5.0	ODBC 1	
----------------	---------	--------	--

SQLSetPos() sets the cursor position in a rowset.

### Syntax

```
SQLRETURN SQLSetPos (
    SQLHSTMT          StatementHandle, /* hstmt */
    SQLSETPOSIROW    RowNumber,       /* irow */
    SQLUSMALLINT     Operation,       /* fOption */
    SQLUSMALLINT     LockType);      /* fLock */
```

### Function arguments

Table 144. SQLSetPos arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle.
SQLSETPOSIROW	<i>RowNumber</i>	input	Position of the row in the rowset on which to perform the operation specified with the <i>Operation</i> argument. If <i>RowNumber</i> is 0, the operation applies to every row in the rowset.  For additional information, see <i>RowNumber</i> argument.
SQLUSMALLINT	<i>Operation</i>	input	Operation to perform: <ul style="list-style-type: none"> <li>• SQL_POSITION</li> <li>• SQL_REFRESH</li> <li>• SQL_UPDATE</li> <li>• SQL_DELETE</li> <li>• SQL_ADD</li> </ul> ODBC also specifies the following operations for backwards compatibility only, which CLI also supports: <ul style="list-style-type: none"> <li>• SQL_ADD</li> </ul> While CLI does support SQL_ADD in SQLSetPos() calls, it is suggested that you use SQLBulkOperations() with the <i>Operation</i> argument set to SQL_ADD.
SQLUSMALLINT	<i>LockType</i>	input	Specifies how to lock the row after performing the operation specified in the <i>Operation</i> argument. <ul style="list-style-type: none"> <li>• SQL_LOCK_NO_CHANGE</li> </ul> ODBC also specifies the following operations which CLI does not support: <ul style="list-style-type: none"> <li>• SQL_LOCK_EXCLUSIVE</li> <li>• SQL_LOCK_UNLOCK</li> </ul> For additional information, see <i>LockType</i> argument.

### Usage

#### RowNumber argument

## SQLSetPos function (CLI) - Set the cursor position in a rowset

The *RowNumber* argument specifies the number of the row in the rowset on which to perform the operation specified by the *Operation* argument. If *RowNumber* is 0, the operation applies to every row in the rowset. *RowNumber* must be a value from 0 to the number of rows in the rowset.

**Note** In the C language, arrays are 0-based, while the *RowNumber* argument is 1-based. For example, to update the fifth row of the rowset, an application modifies the rowset buffers at array index 4, but specifies a *RowNumber* of 5.

All operations position the cursor on the row specified by *RowNumber*. The following operations require a cursor position:

- Positioned update and delete statements.
- Calls to `SQLGetData()`.
- Calls to `SQLSetPos()` with the `SQL_DELETE`, `SQL_REFRESH`, and `SQL_UPDATE` options.

An application can specify a cursor position when it calls `SQLSetPos()`. Generally, it calls `SQLSetPos()` with the `SQL_POSITION` or `SQL_REFRESH` operation to position the cursor before executing a positioned update or delete statement or calling `SQLGetData()`.

### Operation argument

To determine which options are supported by a data source, an application calls `SQLGetInfo()` with one of the following information types, depending on the type of cursor:

- `SQL_DYNAMIC_CURSOR_ATTRIBUTES1`
- `SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1`
- `SQL_KEYSET_CURSOR_ATTRIBUTES1`
- `SQL_STATIC_CURSOR_ATTRIBUTES1`

### SQL\_POSITION

CLI positions the cursor on the row specified by *RowNumber*.

The contents of the row status array pointed to by the `SQL_ATTR_ROW_OPERATION_PTR` statement attribute are ignored for the `SQL_POSITION` *Operation*.

### SQL\_REFRESH

CLI positions the cursor on the row specified by *RowNumber* and refreshes data in the rowset buffers for that row. For more information about how CLI returns data in the rowset buffers, see the descriptions of row-wise and column-wise binding.

`SQLSetPos()` with an *Operation* of `SQL_REFRESH` simply updates the status and content of the rows within the current fetched rowset. This includes refreshing the bookmarks. The data in the buffers is refreshed, but not refetched, so the membership in the rowset is fixed.

A successful refresh with `SQLSetPos()` will not change a row status of `SQL_ROW_DELETED`. Deleted rows within the rowset will continue to be marked as deleted until the next fetch. The rows will disappear at the next fetch if the cursor supports packing (in which case a subsequent `SQLFetch()` or `SQLFetchScroll()` does not return deleted rows).

## SQLSetPos function (CLI) - Set the cursor position in a rowset

A successful refresh with SQLSetPos() will change a row status of SQL\_ROW\_ADDED to SQL\_ROW\_SUCCESS (if the row status array exists).

A refresh with SQLSetPos() will change a row status of SQL\_ROW\_UPDATED to the row's new status (if the row status array exists).

If an error occurs in a SQLSetPos() operation on a row, the row status is set to SQL\_ROW\_ERROR (if the row status array exists).

For a cursor opened with a SQL\_ATTR\_CONCURRENCY statement attribute of SQL\_CONCUR\_ROWVER or SQL\_CONCUR\_VALUES, a refresh with SQLSetPos() will update the optimistic concurrency values used by the data source to detect that the row has changed. This occurs for each row that is refreshed.

The contents of the row status array are ignored for the SQL\_REFRESH Operation.

### SQL\_UPDATE

CLI positions the cursor on the row specified by *RowNumber* and updates the underlying row of data with the values in the rowset buffers (the *TargetValuePtr* argument in SQLBindCol()). It retrieves the lengths of the data from the length/indicator buffers (the *StrLen\_or\_IndPtr* argument in SQLBindCol()). If the length of any column is SQL\_COLUMN\_IGNORE, the column is not updated. After updating the row, the corresponding element of the row status array is updated to SQL\_ROW\_UPDATED or SQL\_ROW\_SUCCESS\_WITH\_INFO (if the row status array exists).

The row operation array pointed to by the SQL\_ATTR\_ROW\_OPERATION\_PTR statement attribute can be used to indicate that a row in the current rowset should be ignored during a bulk update. For more information, see Status and operation arrays.

### SQL\_DELETE

CLI positions the cursor on the row specified by *RowNumber* and deletes the underlying row of data. It changes the corresponding element of the row status array to SQL\_ROW\_DELETED. After the row has been deleted, the following are not valid for the row:

- positioned update and delete statements
- calls to SQLGetData()
- calls to SQLSetPos() with *Operation* set to anything except SQL\_POSITION.

Deleted rows remain visible to static and keyset-driven cursors; however, the entry in the implementation row status array (pointed to by the SQL\_ATTR\_ROW\_STATUS\_PTR statement attribute) for the deleted row is changed to SQL\_ROW\_DELETED.

The row operation array pointed to by the SQL\_ATTR\_ROW\_OPERATION\_PTR statement attribute can be used to indicate that a row in the current rowset should be ignored during a bulk delete. For more information, see Status and operation arrays.

### SQL\_ADD

ODBC also specifies the SQL\_ADD Operation for backwards compatibility only, which CLI also supports. It is suggested, however, that you use SQLBulkOperations() with the *Operation* argument set to SQL\_ADD.



## SQLSetPos function (CLI) - Set the cursor position in a rowset

### LockType argument

The *LockType* argument provides a way for applications to control concurrency. Generally, data sources that support concurrency levels and transactions will only support the SQL\_LOCK\_NO\_CHANGE value of the *LockType* argument.

Although the *LockType* argument is specified for a single statement, the lock accords the same privileges to all statements on the connection. In particular, a lock that is acquired by one statement on a connection can be unlocked by a different statement on the same connection.

ODBC defines the following *LockType* arguments. CLI supports SQL\_LOCK\_NO\_CHANGE. To determine which locks are supported by a data source, an application calls SQLGetInfo() with the SQL\_LOCK\_TYPES information type.

Table 145. Operation values

LockType argument	Lock type
SQL_LOCK_NO_CHANGE	Ensures that the row is in the same locked or unlocked state as it was before SQLSetPos() was called. This value of <i>LockType</i> allows data sources that do not support explicit row-level locking to use whatever locking is required by the current concurrency and transaction isolation levels.
SQL_LOCK_EXCLUSIVE	Not supported by CLI. Locks the row exclusively.
SQL_LOCK_UNLOCK	Not supported by CLI. Unlocks the row.

### Status and operation arrays

The following status and operation arrays are used when calling SQLSetPos():

- The row status array (as pointed to by the SQL\_DESC\_ARRAY\_STATUS\_PTR field in the IRD and the SQL\_ATTR\_ROW\_STATUS\_ARRAY statement attribute) contains status values for each row of data in the rowset. The status values are set in this array after a call to SQLFetch(), SQLFetchScroll(), or SQLSetPos(). This array is pointed to by the SQL\_ATTR\_ROW\_STATUS\_PTR statement attribute.
- The row operation array (as pointed to by the SQL\_DESC\_ARRAY\_STATUS\_PTR field in the ARD and the SQL\_ATTR\_ROW\_OPERATION\_ARRAY statement attribute) contains a value for each row in the rowset that indicates whether a call to SQLSetPos() for a bulk operation is ignored or performed. Each element in the array is set to either SQL\_ROW\_PROCEED (the default) or SQL\_ROW\_IGNORE. This array is pointed to by the SQL\_ATTR\_ROW\_OPERATION\_PTR statement attribute.

The number of elements in the status and operation arrays must equal the number of rows in the rowset (as defined by the SQL\_ATTR\_ROW\_ARRAY\_SIZE statement attribute).

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_NEED\_DATA
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## SQLSetPos function (CLI) - Set the cursor position in a rowset

### Diagnostics

Table 146. SQLSetPos SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The <i>Operation</i> argument was SQL_REFRESH, and string or binary data returned for a column or columns with a data type of SQL_C_CHAR or SQL_C_BINARY resulted in the truncation of non-blank character or non-NULL binary data
01S01	Error in row.	<p>The <i>RowNumber</i> argument was 0 and an error occurred in one or more rows while performing the operation specified with the <i>Operation</i> argument.</p> <p>(SQL_SUCCESS_WITH_INFO is returned if an error occurs on one or more, but not all, rows of a multirow operation, and SQL_ERROR is returned if an error occurs on a single-row operation.)</p>
01S07	Fractional truncation.	The <i>Operation</i> argument was SQL_REFRESH, the data type of the application buffer was not SQL_C_CHAR or SQL_C_BINARY, and the data returned to application buffers for one or more columns was truncated. For numeric data types, the fractional part of the number was truncated. For time and timestamp data types, the fractional portion of the time was truncated.
07006	Invalid conversion.	The data value of a column in the result set could not be converted to the data type specified by <i>TargetType</i> in the call to SQLBindCol().
07009	Invalid descriptor index.	The argument <i>Operation</i> was SQL_REFRESH or SQL_UPDATE and a column was bound with a column number greater than the number of columns in the result set or a column number less than 0.
21S02	Degrees of derived table does not match column list.	The argument <i>Operation</i> was SQL_UPDATE and no columns were updateable because all columns were either unbound, read-only, or the value in the bound length/indicator buffer was SQL_COLUMN_IGNORE.
22001	String data right truncation.	The assignment of a character or binary value to a column resulted in the truncation of non-blank (for characters) or non-null (for binary) characters or bytes.
22003	Numeric value out of range.	<p>The argument <i>Operation</i> was SQL_UPDATE and the assignment of a numeric value to a column in the result set caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>The argument <i>Operation</i> was SQL_REFRESH, and returning the numeric value for one or more bound columns would have caused a loss of significant digits.</p>
22007	Invalid datetime format.	<p>The argument <i>Operation</i> was SQL_UPDATE, and the assignment of a date or timestamp value to a column in the result set caused the year, month, or day field to be out of range.</p> <p>The argument <i>Operation</i> was SQL_REFRESH, and returning the date or timestamp value for one or more bound columns would have caused the year, month, or day field to be out of range.</p>

## SQLSetPos function (CLI) - Set the cursor position in a rowset

Table 146. SQLSetPos SQLSTATEs (continued)

SQLSTATE	Description	Explanation
22008	Datetime field overflow.	<p>The <i>Operation</i> argument was SQL_UPDATE, and the performance of datetime arithmetic on data being sent to a column in the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar.</p> <p>The <i>Operation</i> argument was SQL_REFRESH, and the performance of datetime arithmetic on data being retrieved from the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar.</p>
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY010	Function sequence error.	<p>The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect(), SQLExecute(), or a catalog function.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>SQLExecute(), SQLExecDirect(), or SQLSetPos() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>An ODBC 2.0 application called SQLSetPos() for a <i>StatementHandle</i> before SQLFetchScroll() was called or after SQLFetch() was called, and before SQLFreeStmt() was called with the SQL_CLOSE option.</p>
HY011	Operation invalid at this time.	An ODBC 2.0 application set the SQL_ATTR_ROW_STATUS_PTR statement attribute; then SQLSetPos() was called before SQLFetch(), SQLFetchScroll(), or SQLExtendedFetch() was called.

## SQLSetPos function (CLI) - Set the cursor position in a rowset

Table 146. SQLSetPos SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	<p>The <i>Operation</i> argument was SQL_ADD, SQL_UPDATE, or SQL_UPDATE_BY_BOOKMARK, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The <i>Operation</i> argument was SQL_ADD, SQL_UPDATE, or SQL_UPDATE_BY_BOOKMARK, a data value was not a null pointer, and the column length value was less than 0, but not equal to SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>A value in a length/indicator buffer was SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a other, data-source-specific data type; and the SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo() was Y.</p>
HY092	Option type out of range.	The <i>Operation</i> argument was SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK, or SQL_REFRESH_BY_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
HY107	Row value out of range.	The value specified for the argument <i>RowNumber</i> was greater than the number of rows in the rowset.
HY109	Invalid cursor position.	<p>The cursor associated with the <i>StatementHandle</i> was defined as forward only, so the cursor could not be positioned within the rowset. See the description for the SQL_ATTR_CURSOR_TYPE attribute in SQLSetStmtAttr().</p> <p>The <i>Operation</i> argument was SQL_UPDATE, SQL_DELETE, or SQL_REFRESH, and the row identified by the <i>RowNumber</i> argument had been deleted or had not be fetched.</p> <p>The <i>RowNumber</i> argument was 0 and the <i>Operation</i> argument was SQL_POSITION.</p>
HYC00	Driver not capable.	CLI or the data source does not support the operation requested in the <i>Operation</i> argument or the <i>LockType</i> argument.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr() with an <i>Attribute</i> of SQL_ATTR_QUERY_TIMEOUT.

### Restrictions

None.

### Example

```
/* set the cursor position in a rowset */
cliRC = SQLSetPos(hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
```

## SQLSetStmtAttr function (CLI) - Set options related to a statement

### Purpose

Specification:	CLI 5.0	ODBC 3.0	ISO CLI
----------------	---------	----------	---------

SQLSetStmtAttr() sets options related to a statement. To set an option for all statements associated with a specific connection, an application can call SQLSetConnectAttr().

Refer to the CLI statement attributes list for all available statement attributes.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSetStmtAttrW(). Refer to “Unicode functions (CLI)” on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLSetStmtAttr (
    SQLHSTMT      StatementHandle, /* hstmt */
    SQLINTEGER    Attribute,       /* fOption */
    SQLPOINTER    ValuePtr,       /* pvParam */
    SQLINTEGER    StringLength);  /* fStrLen */
```

### Function arguments

Table 147. SQLSetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	input	Statement handle.
SQLINTEGER	<i>Attribute</i>	input	Option to set, described in the CLI statement attributes list.
SQLPOINTER	<i>ValuePtr</i>	input	<p>Pointer to the value to be associated with <i>Attribute</i>.</p> <p>If <i>Attribute</i> is an ODBC-defined attribute, the application might need to qualify the attribute value in <i>ValuePtr</i> by setting the <i>StringLength</i> attribute as described in the <i>StringLength</i> description.</p> <p>If <i>Attribute</i> is a CLI attribute, the application should always qualify the attribute value in <i>ValuePtr</i> by setting the <i>StringLength</i> attribute as described in the <i>StringLength</i> description.</p> <p><b>Note:</b> If <i>Attribute</i> is an ODBC attribute, <i>ValuePtr</i> can, depending on the attribute, be set to an unsigned integer. If <i>Attribute</i> is a CLI attribute, <i>ValuePtr</i> can, depending on the attribute, be set to a signed integer. If <i>ValuePtr</i> is set to a signed negative integer and an unsigned integer is expected, <i>ValuePtr</i> might be treated as a large unsigned integer by CLI without warning. Alternatively, CLI might return an error (SQLSTATE HY024).</p>

## SQLSetStmtAttr function (CLI) - Set options related to a statement

Table 147. SQLSetStmtAttr arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>StringLength</i>	input	<p>If <i>Attribute</i> is an ODBC attribute, the application might need to qualify the attribute by setting <i>StringLength</i> to the following values:</p> <ul style="list-style-type: none"> <li>• If <i>ValuePtr</i> points to a character string or a binary buffer, <i>StringLength</i> should be the length of <i>*ValuePtr</i>. For character string data, <i>StringLength</i> should contain the number of bytes in the string.</li> <li>• If <i>ValuePtr</i> is a pointer, but not to a string or binary buffer, then <i>StringLength</i> should have the value SQL_IS_POINTER.</li> <li>• If <i>ValuePtr</i> points to an unsigned integer, the <i>StringLength</i> attribute is ignored.</li> </ul> <p>If <i>Attribute</i> is a CLI attribute, the application must qualify the attribute by setting <i>StringLength</i> to the following values:</p> <ul style="list-style-type: none"> <li>• If <i>ValuePtr</i> is a pointer to a character string, then <i>StringLength</i> is the number of bytes needed to store the string or SQL_NTS.</li> <li>• If <i>ValuePtr</i> is a pointer to a binary buffer, then the application should place the result of the SQL_LEN_BINARY_ATTR (length) macro in <i>StringLength</i>. This places a negative value in <i>StringLength</i>.</li> <li>• If <i>ValuePtr</i> contains a fixed-length value, then <i>StringLength</i> is either SQL_IS_INTEGER or SQL_IS_UIINTEGER, as appropriate.</li> <li>• If <i>ValuePtr</i> is a pointer to a value other than a character string, a binary string, or a fixed-length value, then <i>StringLength</i> should have the value SQL_IS_POINTER.</li> </ul>

### Usage

Statement attributes for a statement remain in effect until they are changed by another call to SQLSetStmtAttr() or the statement is dropped by calling SQLFreeHandle(). Calling SQLFreeStmt() with the SQL\_CLOSE, SQL\_UNBIND, or SQL\_RESET\_PARAMS options does not reset statement attributes.

Some statement attributes support substitution of a similar value if the data source does not support the value specified in *\*ValuePtr*. In such cases, CLI returns SQL\_SUCCESS\_WITH\_INFO and SQLSTATE 01S02 (Option value changed). For example, CLI supports a pure keyset cursor. As a result, CLI does not allow applications to change the default value of the SQL\_ATTR\_KEYSET\_SIZE attribute. Instead, CLI substitutes SQL\_KEYSET\_SIZE\_DEFAULT for all other values that might be supplied in the *\*ValuePtr* argument and returns SQL\_SUCCESS\_WITH\_INFO. To determine the substituted value, an application calls SQLGetStmtAttr().

The format of information set with *ValuePtr* depends on the specified *Attribute*. SQLSetStmtAttr() accepts attribute information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of information returned in SQLGetStmtAttr() reflects what was specified in

## SQLSetStmtAttr function (CLI) - Set options related to a statement

SQLSetStmtAttr(). For example, character strings pointed to by the *ValuePtr* argument of SQLSetStmtAttr() have a length of *StringLength*, and this is the value that would be returned by SQLGetStmtAttr().

### Setting statement attributes by setting descriptors

Many statement attributes also corresponding to a header field of one or more descriptors. These attributes can be set not only by a call to SQLSetStmtAttr(), but also by a call to SQLSetDescField(). Setting these options by a call to SQLSetStmtAttr(), rather than SQLSetDescField(), has the advantage that a descriptor handle does not have to be fetched.

**Note:** Calling SQLSetStmtAttr() for one statement can affect other statements. This occurs when the application parameter descriptor (APD) or application row descriptor (ARD) associated with the statement is explicitly allocated and is also associated with other statements. Because SQLSetStmtAttr() modifies the APD or ARD, the modifications apply to all statements with which this descriptor is associated. If this is not the desired behavior, the application should dissociate this descriptor from the other statement (by calling SQLSetStmtAttr() to set the SQL\_ATTR\_APP\_ROW\_DESC or SQL\_ATTR\_APP\_PARAM\_DESC field to a different descriptor handle) before calling SQLSetStmtAttr() again.

When a statement attribute that is also a descriptor field is set by a call to SQLSetStmtAttr(), the corresponding field in the descriptor that is associated with the statement is also set. The field is set only for the applicable descriptors that are currently associated with the statement identified by the *StatementHandle* argument, and the attribute setting does not affect any descriptors that might be associated with that statement in the future. When a descriptor field that is also a statement attribute is set by a call to SQLSetDescField(), the corresponding statement attribute is also set.

Statement attributes determine which descriptors a statement handle is associated with. When a statement is allocated (see SQLAllocHandle()), four descriptor handles are automatically allocated and associated with the statement. Explicitly allocated descriptor handles can be associated with the statement by calling SQLAllocHandle() with a *HandleType* of SQL\_HANDLE\_DESC to allocate a descriptor handle, then calling SQLSetStmtAttr() to associate the descriptor handle with the statement.

The following statement attributes correspond to descriptor header fields:

Table 148. Statement attributes

Statement attribute	Header field	Descriptor
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

## SQLSetStmtAttr function (CLI) - Set options related to a statement

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 149. SQLSetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed.	CLI did not support the value specified in <i>*ValuePtr</i> , or the value specified in <i>*ValuePtr</i> was invalid because of SQL constraints or requirements, so CLI substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The communication link between CLI and the data source to which it was connected failed before the function completed processing.
24000	Invalid cursor state.	The <i>Attribute</i> was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS and the cursor was open.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	A null pointer was passed for <i>ValuePtr</i> and the value in <i>*ValuePtr</i> was a string attribute.
HY010	Function sequence error.	An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called.  SQLExecute() or SQLExecDirect() was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY011	Operation invalid at this time.	The <i>Attribute</i> was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS and the statement was prepared.
HY017	Invalid use of an automatically allocated descriptor handle.	The <i>Attribute</i> argument was SQL_ATTR_IMP_ROW_DESC or SQL_ATTR_IMP_PARAM_DESC. The <i>Attribute</i> argument was SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and the value in <i>*ValuePtr</i> was an implicitly allocated descriptor handle.
HY024	Invalid attribute value.	Given the specified <i>Attribute</i> value, an invalid value was specified in <i>*ValuePtr</i> . (CLI returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE.)
HY090	Invalid string or buffer length.	The <i>StringLength</i> argument was less than 0, but was not SQL_NTS.



## SQLSetStmtAttr function (CLI) - Set options related to a statement

Table 149. SQLSetStmtAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Option type out of range.	The value specified for the argument <i>Attribute</i> was not valid for this version of CLI.
HYC00	Driver not capable.	The value specified for the argument <i>Attribute</i> was a valid connection or statement attribute for the version of the CLI driver, but was not supported by the data source.

### Restrictions

None.

### Example

```
/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                      SQL_ATTR_ROW_ARRAY_SIZE,
                      (SQLPOINTER)ROWSET_SIZE,
                      0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                      SQL_ATTR_ROW_BIND_TYPE,
                      SQL_BIND_BY_COLUMN,
                      0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                      SQL_ATTR_ROWS_FETCHED_PTR,
                      &rowsFetchedNb,
                      0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
```

## SQLSetStmtOption function (CLI) - Set statement option

### Deprecated

#### Note:

In ODBC 3.0, SQLSetStmtOption() has been deprecated and replaced with SQLSetStmtAttr().

Although this version of CLI continues to support SQLSetStmtOption(), use SQLSetStmtAttr() in your CLI programs so that they conform to the latest standards.

**Note:** This deprecated function cannot be used in a 64-bit environment.

### Migrating to the new function

The statement:

```
SQLSetStmtOption(
    hstmt,
    SQL_ROWSET_SIZE,
    RowSetSize);
```

## SQLSetStmtOption function (CLI) - Set statement option

for example, would be rewritten using the new function as:

```
SQLSetStmtAttr(  
    hstmt,  
    SQL_ATTR_ROW_ARRAY_SIZE,  
    (SQLPOINTER) RowSetSize,  
    0);
```

---

## SQLSpecialColumns function (CLI) - Get special (row identifier) columns

### Purpose

Specification:	DB2 Call Level Interface 2.1	ODBC 1.0	
----------------	------------------------------	----------	--

SQLSpecialColumns() returns unique row identifier information (for example, the primary key or unique index) for a table. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

**Unicode equivalent:** This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSpecialColumnsW(). Refer to "Unicode functions (CLI)" on page 5 for information on ANSI to Unicode function mappings.

### Syntax

```
SQLRETURN SQLSpecialColumns(  
    SQLHSTMT StatementHandle, /* hstmt */  
    SQLUSMALLINT IdentifierType, /* fColType */  
    SQLCHAR *CatalogName, /* szCatalogName */  
    SQLSMALLINT NameLength1, /* cbCatalogName */  
    SQLCHAR *SchemaName, /* szSchemaName */  
    SQLSMALLINT NameLength2, /* cbSchemaName */  
    SQLCHAR *TableName, /* szTableName */  
    SQLSMALLINT NameLength3, /* cbTableName */  
    SQLUSMALLINT Scope, /* fScope */  
    SQLUSMALLINT Nullable); /* fNullable */
```

### Function arguments

Table 150. SQLSpecialColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle
SQLUSMALLINT	<i>IdentifierType</i>	Input	Type of unique row identifier to return. Only the following type is supported: <ul style="list-style-type: none"><li>SQL_BEST_ROWID</li></ul> Returns the optimal set of column(s) which can uniquely identify any row in the specified table.  <b>Note:</b> For compatibility with ODBC applications, SQL_ROWVER is also recognized, but not supported; therefore, if SQL_ROWVER is specified, an empty result will be returned.

## SQLSpecialColumns function (CLI) - Get special (row identifier) columns

Table 150. SQLSpecialColumns arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>CatalogName</i> , or SQL_NTS if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	Schema qualifier of the specified table.
SQLSMALLINT	<i>NameLength2</i>	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>SchemaName</i> , or SQL_NTS if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	Table name.
SQLSMALLINT	<i>NameLength3</i>	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>TableName</i> , or SQL_NTS if <i>TableName</i> is null-terminated.
SQLUSMALLINT	<i>Scope</i>	Input	<p>Minimum required duration for which the unique row identifier will be valid.</p> <p><i>Scope</i> must be one of the following:</p> <ul style="list-style-type: none"> <li>• SQL_SCOPE_CURROW: The row identifier is guaranteed to be valid only while positioned on that row. A later re-select using the same row identifier values might not return a row if the row was updated or deleted by another transaction.</li> <li>• SQL_SCOPE_TRANSACTION: The row identifier is guaranteed to be valid for the duration of the current transaction.</li> <li>• SQL_SCOPE_SESSION: The row identifier is guaranteed to be valid for the duration of the connection.</li> </ul> <p>The duration over which a row identifier value is guaranteed to be valid depends on the current transaction isolation level.</p>
SQLUSMALLINT	<i>Nullable</i>	Input	<p>Determines whether to return special columns that can have a NULL value.</p> <p>Must be one of the following:</p> <ul style="list-style-type: none"> <li>• SQL_NO_NULLS - The row identifier column set returned cannot have any NULL values.</li> <li>• SQL_NULLABLE - The row identifier column set returned might include columns where NULL values are permitted.</li> </ul>

## SQLSpecialColumns function (CLI) - Get special (row identifier) columns

### Usage

If multiple ways exist to uniquely identify any row in a table (for example, if there are multiple unique indexes on the specified table), then DB2 Call Level Interface will return the *best* set of row identifier column set based on its internal criterion.

If the schema qualifier argument associated with a table name is not specified, then the schema name defaults to the one currently in effect for the current connection.

If there is no column set which allows any row in the table to be uniquely identified, an empty result set is returned.

The unique row identifier information is returned in the form of a result set where each column of the row identifier is represented by one row in the result set. Columns returned by SQLSpecialColumns shows the order of the columns in the result set returned by SQLSpecialColumns(), sorted by SCOPE.

Since calls to SQLSpecialColumns() in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

Call SQLGetInfo() with the SQL\_MAX\_COLUMN\_NAME\_LEN to determine the actual length of the COLUMN\_NAME column supported by the connected DBMS.

Although new columns might be added and the names of the columns changed in future releases, the position of the current columns will not change.

**Note:** IDS data servers have a virtual column named ROWID for every non-fragmented table. The SQLSpecialColumns() function will return information on the ROWID column when accessing IDS data servers.

### Columns returned by SQLSpecialColumns

#### Column 1 SCOPE (SMALLINT)

The duration for which the name in COLUMN\_NAME is guaranteed to point to the same row. Valid values are the same as for the *Scope* argument: Actual scope of the row identifier. Contains one of the following values:

- SQL\_SCOPE\_CURROW
- SQL\_SCOPE\_TRANSACTION
- SQL\_SCOPE\_SESSION

Refer to *Scope* in Table 150 on page 294 for a description of each value.

#### Column 2 COLUMN\_NAME (VARCHAR(128) not NULL)

Name of the column that is (or is part of) the table's primary key.

#### Column 3 DATA\_TYPE (SMALLINT not NULL)

SQL data type of the column.

#### Column 4 TYPE\_NAME (VARCHAR(128) not NULL)

DBMS character string representation of the name associated with DATA\_TYPE column value.

#### Column 5 COLUMN\_SIZE (INTEGER)

If the DATA\_TYPE column value denotes a character or binary string, then this column contains the maximum length in bytes; if it is a graphic (DBCS) string, this is the number of double byte characters for the parameter.

## SQLSpecialColumns function (CLI) - Get special (row identifier) columns

For date, time, timestamp data types, this is the total number of SQLCHAR or SQLWCHAR elements required to display the value when converted to character.

For numeric data types, this is either the total number of digits, or the total number of bits allowed in the column, depending on the value in the NUM\_PREC\_RADIX column in the result set.

Refer to the table of data type precision.

### Column 6 BUFFER\_LENGTH (INTEGER)

The maximum number of bytes for the associated C buffer to store data from this column if SQL\_C\_DEFAULT were specified on the SQLBindCol(), SQLGetData() and SQLBindParameter() calls. This length does not include any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

Refer to the table of data type length.

### Column 7 DECIMAL\_DIGITS (SMALLINT)

The scale of the column. NULL is returned for data types where scale is not applicable. Refer to the table of data type scale.

### Column 8 PSEUDO\_COLUMN (SMALLINT)

Indicates whether or not the column is a pseudo-column DB2 Call Level Interface will only return:

- SQL\_PC\_NOT\_PSEUDO

DB2 DBMSs do not support pseudo columns. ODBC applications might receive the following values from other non-IBM RDBMS servers:

- SQL\_PC\_UNKNOWN
- SQL\_PC\_PSEUDO

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 151. SQLSpecialColumns SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel() was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	<i>TableName</i> is null.

## SQLSpecialColumns function (CLI) - Get special (row identifier) columns

Table 151. SQLSpecialColumns SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	<p>The value of one of the length arguments was less than 0, but not equal to SQL_NTS.</p> <p>The value of one of the length arguments exceeded the maximum length supported by the DBMS for that qualifier or name.</p>
HY097	Column type out of range.	An invalid <i>IdentifierType</i> value was specified.
HY098	Scope type out of range.	An invalid <i>Scope</i> value was specified.
HY099	Nullable type out of range.	An invalid <i>Nullable</i> values was specified.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* get special columns */
cliRC = SQLSpecialColumns(hstmt,
                        SQL_BEST_ROWID,
                        NULL,
                        0,
                        tbSchema,
                        SQL_NTS,
                        tbName,
                        SQL_NTS,
                        SQL_SCOPE_CURROW,
                        SQL_NULLABLE);
```

## SQLStatistics function (CLI) - Get index and statistics information for a base table

The SQLStatistics() function retrieves index information for a given table.

The SQLStatistics() function also returns the cardinality and the number of pages that are associated with the table and the indexes on the table.

## SQLStatistics function (CLI) - Get index and statistics information for a base table

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

The information is returned in a result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLStatisticsW()`. For information about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLStatistics (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *CatalogName, /* szCatalogName */
    SQLSMALLINT NameLength1, /* cbCatalogName */
    SQLCHAR *SchemaName, /* szSchemaName */
    SQLSMALLINT NameLength2, /* cbSchemaName */
    SQLCHAR *TableName, /* szTableName */
    SQLSMALLINT NameLength3, /* cbTableName */
    SQLUSMALLINT Unique, /* fUnique */
    SQLUSMALLINT Reserved); /* fAccuracy */
```

### Function arguments

Table 152. SQLStatistics arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and <code>SQL_SUCCESS</code> is returned. Otherwise, this is a valid filter for DBMSs that supports 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or <code>SQL_NTS</code> if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	The schema qualifier of the specified table.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or <code>SQL_NTS</code> if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	The table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableName</i> , or <code>SQL_NTS</code> if <i>TableName</i> is null-terminated.
SQLUSMALLINT	<i>Unique</i>	Input	The type of index information to return: <ul style="list-style-type: none"> <li>• <code>SQL_INDEX_UNIQUE</code> Only unique indexes are returned.</li> <li>• <code>SQL_INDEX_ALL</code> All indexes are returned.</li> </ul>

## SQLStatistics function (CLI) - Get index and statistics information for a base table

Table 152. SQLStatistics arguments (continued)

Data type	Argument	Use	Description
SQLUSMALLINT	<i>Reserved</i>	Input	Indicates whether the CARDINALITY and PAGES columns in the result set contain the most current information: <ul style="list-style-type: none"><li>• SQL_ENSURE : This value is reserved for future use, when the application requests the most up-to-date statistics information. <b>New applications should not use this value.</b> Existing applications that specify this value will receive the same results as SQL_QUICK.</li><li>• SQL_QUICK : Statistics which are readily available at the server are returned. No attempt is made to ensure that the values are current.</li></ul>

### Usage

The SQLStatistics() function returns two types of information:

- Statistics information for the table (if it is available):
  - if the TYPE column of the result set is set to SQL\_TABLE\_STAT, the number of rows in the table, and the number of pages that are used to store the table are returned.
  - if the TYPE column of the result set indicates an index, the number of unique values in the index, and the number of pages that are used to store the indexes are returned.
- Information about each index, where each index column is represented by one row of the result set. The result set columns are described in Columns returned by SQLStatistics. The rows in the result set are ordered by NON\_UNIQUE, TYPE, INDEX\_QUALIFIER, INDEX\_NAME and KEY\_SEQ columns.

In many cases, calls to the SQLStatistics() function map to a complex and thus expensive query against the system catalog, so you should use the calls sparingly, and save the results rather than repeating calls.

If the schema qualifier argument that is associated with a table name is not specified, the schema name defaults to the argument that is in effect for the current connection.

Call the SQLGetInfo() function with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_OWNER\_SCHEMA\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

In Version 9.7 Fix Pack 5, you can specify \*ALL as a value in the *SchemaName* to resolve unqualified stored procedure calls, or to find libraries in catalog API calls. CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.



## SQLStatistics function (CLI) - Get index and statistics information for a base table

### Columns returned by SQLStatistics

#### Column 1 TABLE\_CAT (VARCHAR(128))

The catalog name of the table for which the index applies. The value is NULL if this table does not have catalogs.

#### Column 2 TABLE\_SCHEM (VARCHAR(128))

The name of the schema containing TABLE\_NAME.

#### Column 3 TABLE\_NAME (VARCHAR(128) not NULL)

The name of the table.

#### Column 4 NON\_UNIQUE (SMALLINT)

Indicates whether the index prohibits duplicate values:

- SQL\_TRUE is returned if the index allows duplicate values.
- SQL\_FALSE is returned if the index values must be unique.
- NULL is returned if the TYPE column indicates that this row is SQL\_TABLE\_STAT (statistics information about the table).

#### Column 5 INDEX\_QUALIFIER (VARCHAR(128))

The string that is used to qualify the index name in the DROP INDEX statement. Appending a period (.) plus the INDEX\_NAME results in a full specification of the index.

#### Column 6 INDEX\_NAME (VARCHAR(128))

The name of the index. If the TYPE column has the value SQL\_TABLE\_STAT, this column has the value NULL.

#### Column 7 TYPE (SMALLINT not NULL)

Indicates the type of information that is contained in this row of the result set:

- SQL\_TABLE\_STAT indicates that this row contains statistics information about the table.
- SQL\_INDEX\_CLUSTERED indicates that this row contains information about an index, and the index type is a clustered index.
- SQL\_INDEX\_HASHED indicates that this row contains information about an index, and the index type is a hashed index.
- SQL\_INDEX\_OTHER indicates that this row contains information about an index that is not clustered or hashed

#### Column 8 ORDINAL\_POSITION (SMALLINT)

The ordinal position of the column in the index whose name is given in the INDEX\_NAME column. A NULL value is returned for this column if the TYPE column has the value of SQL\_TABLE\_STAT.

#### Column 9 COLUMN\_NAME (VARCHAR(128))

The name of the column in the index. A NULL value is returned for this column if the TYPE column has the value of SQL\_TABLE\_STAT.

#### Column 10 ASC\_OR\_DESC (CHAR(1))

The sort sequence for the column; "A" for ascending, or "D" for descending. A NULL value is returned if the value in the TYPE column is SQL\_TABLE\_STAT.

#### Column 11 CARDINALITY (INTEGER)

- If the TYPE column contains the value SQL\_TABLE\_STAT, this column contains the number of rows that are in the table.
- If the TYPE column value is not SQL\_TABLE\_STAT, this column contains the number of unique values that are in the index.
- A NULL value is returned if information is not available from the DBMS.

## SQLStatistics function (CLI) - Get index and statistics information for a base table

### Column 12 PAGES (INTEGER)

- If the TYPE column contains the value SQL\_TABLE\_STAT, this column contains the number of pages that are used to store the table.
- If the TYPE column value is not SQL\_TABLE\_STAT, this column contains the number of pages that are used to store the indexes.
- A NULL value is returned if information is not available from the DBMS.

### Column 13 FILTER\_CONDITION (VARCHAR(128))

If the index is a filtered index, this is the filter condition. Because DB2 servers do not support filtered indexes, NULL is always returned. NULL is also returned if TYPE is SQL\_TABLE\_STAT.

For the row in the result set that contains table statistics (TYPE is set to SQL\_TABLE\_STAT), the columns values of NON\_UNIQUE, INDEX\_QUALIFIER, INDEX\_NAME, ORDINAL\_POSITION, COLUMN\_NAME, and ASC\_OR\_DESC are set to NULL. If the CARDINALITY or PAGES information cannot be determined, then NULL is returned for those columns.

**Note:** An application can check the SQLERRD(3) and SQLERRD(4) fields of the SQLCA to gather some statistics on a table. However, the accuracy of the information that is returned in those fields depends on many factors, such as the use of parameter markers and expressions within the statement. The main factor that you can control is the accuracy of the database statistics. For example, for DB2 Database for Linux, UNIX, and Windows, the last time the RUNSTATS command was run. Therefore, the statistics information that is returned by SQLStatistics() is often more consistent and reliable than the statistics information that is contained in the SQLCA fields that were previously explained.

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 153. SQLStatistics SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, SQLCancel () was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	<i>TableName</i> is null.

## SQLStatistics function (CLI) - Get index and statistics information for a base table

Table 153. SQLStatistics SQLSTATES (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	<p>The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.</p> <p>The valid of one of the name-length arguments exceeded the maximum value that is supported for that data source. The maximum supported value can be obtained by calling the SQLGetInfo() function.</p>
HY100	Uniqueness option type out of range.	An invalid <i>Unique</i> value was specified.
HY101	Accuracy option type out of range.	An invalid <i>Reserved</i> value was specified.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* get index and statistics information for a base table */
cliRC = SQLStatistics(hstmt,
                    NULL,
                    0,
                    tbSchema,
                    SQL_NTS,
                    tbName,
                    SQL_NTS,
                    SQL_INDEX_UNIQUE,
                    SQL_QUICK);
```

## SQLTablePrivileges function (CLI) - Get privileges associated with a table

The SQLTablePrivileges() function returns a list of tables and associated privileges for each table.

The information is returned in an SQL result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

## SQLTablePrivileges function (CLI) - Get privileges associated with a table

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLTablePrivilegesW()`. For details about ANSI to Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLTablePrivileges (
    SQLHSTMT      StatementHandle, /* hstmt */
    SQLCHAR       *CatalogName,   /* *szCatalogName */
    SQLSMALLINT   NameLength1,   /* cbCatalogName */
    SQLCHAR       *SchemaName,    /* *szSchemaName */
    SQLSMALLINT   NameLength2,   /* cbSchemaName */
    SQLCHAR       *TableName,     /* *szTableName */
    SQLSMALLINT   NameLength3);  /* cbTableName */
```

### Function arguments

Table 154. *SQLTablePrivileges* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	The catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and <i>PKCatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and <code>SQL_SUCCESS</code> is returned. Otherwise, this is a valid filter for DBMSs that supports 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or <code>SQL_NTS</code> if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by schema name.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or <code>SQL_NTS</code> if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableName</i> , or <code>SQL_NTS</code> if <i>TableName</i> is null-terminated.

Note that the *SchemaName* and *TableName* input arguments accept search patterns.

### Usage

The results are returned as a standard result set that contain the columns that are listed in the succeeding table. The result set is ordered by the `TABLE_CAT`, `TABLE_SCHEM`, `TABLE_NAME`, and `PRIVILEGE` columns. If multiple privileges are associated with any given table, each privilege is returned as a separate row.

## SQLTablePrivileges function (CLI) - Get privileges associated with a table

The granularity of each privilege that is reported here might apply at the column level. For example, for some data sources, if you can update a table, you can also update every column in that table. For other data sources, the application must call `SQLColumnPrivileges()` to discover if the individual columns have the same table privileges.

In many cases, calls to the `SQLTablePrivileges()` function map to a complex and thus expensive query against the system catalog, so you should use the calls sparingly, and save the results rather than repeating the calls.

Sometimes, an application calls the function and no attempt is made to restrict the result set that is returned. For some data sources that contain a large quantity of tables, views, and aliases for example, this scenario maps to an extremely large result set and very long retrieval times. In order to help reduce the long retrieval times, you can specify the `SchemaList` configuration keyword in the CLI initialization file to help restrict the result set when the application has supplied a null pointer for the `SchemaName`. If the application specifies a `SchemaName` string, the `SchemaList` keyword is still used to restrict the output. Therefore, if the schema name that is supplied is not in the `SchemaList` string, the result is an empty result set.

Call `SQLGetInfo()` with the `SQL_MAX_CATALOG_NAME_LEN`, `SQL_MAX_OWNER_SCHEMA_LEN`, `SQL_MAX_TABLE_NAME_LEN`, and `SQL_MAX_COLUMN_NAME_LEN` to determine respectively the actual lengths of the `TABLE_CAT`, `TABLE_SCHEM`, `TABLE_NAME`, and `COLUMN_NAME` columns that are supported by the connected DBMS.

In Version 9.7 Fix Pack 5, you can specify `*ALL` or `*USRLIBL` as values in the `SchemaName` to resolve unqualified stored procedure calls or to find libraries in catalog API calls. If you specify `*ALL`, CLI searches on all existing schemas in the connected database. You are not required to specify `*ALL`, as this behavior is the default in CLI. For IBM DB2 for IBM i servers, if you specify `*USRLIBL`, CLI searches on the current libraries of the server job. For other DB2 servers, `*USRLIBL` does not have a special meaning and CLI searches using `*USRLIBL` as a pattern. Alternatively, you can set the `SchemaFilter` IBM Data Server Driver configuration keyword or the `Schema List` CLI/ODBC configuration keyword to `*ALL` or `*USRLIBL`.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns returned by SQLTablePrivileges

#### Column 1 TABLE\_CAT (VARCHAR(128))

The name of the catalog table. The value is NULL if this table does not have catalogs.

#### Column 2 TABLE\_SCHEM (VARCHAR(128))

The name of the schema that contains TABLE\_NAME.

#### Column 3 TABLE\_NAME (VARCHAR(128) not NULL)

The name of the table.

#### Column 4 GRANTOR (VARCHAR(128))

The authorization ID of the user who granted the privilege.

#### Column 5 GRANTEE (VARCHAR(128))

The authorization ID of the user to whom the privilege is granted.

## SQLTablePrivileges function (CLI) - Get privileges associated with a table

### Column 6 PRIVILEGE (VARCHAR(128))

The table privilege. This privilege can be one of the listed strings:

- ALTER
- CONTROL
- DELETE
- INDEX
- INSERT
- REFERENCES
- SELECT
- UPDATE

### Column 7 IS\_GRANTABLE (VARCHAR(3))

Indicates whether the grantee is permitted to grant the privilege to other users.

The IS\_GRANTABLE value can be "YES", "NO," or NULL.

**Note:** The column names that are used by CLI follow the X/Open CLI CAE specification style. The column types, contents, and order are identical to those that are defined for the SQLProcedures() result set in ODBC.

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 155. SQLTablePrivileges SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .

## SQLTablePrivileges function (CLI) - Get privileges associated with a table

Table 155. SQLTablePrivileges SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	<p>The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.</p> <p>The function was called during a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	<p>The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.</p> <p>The valid value of one of the name-length arguments exceeded the maximum value that is supported for that data source. You can obtain the maximum supported value by calling the SQLGetInfo() function.</p>
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. You can set the timeout period by using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

### Restrictions

None.

### Example

```
/* get privileges associated with a table */
cliRC = SQLTablePrivileges(hstmt,
                           NULL,
                           0,
                           tbSchemaPattern,
                           SQL_NTS,
                           tbNamePattern,
                           SQL_NTS);
```

## SQLTables function (CLI) - Get table information

The SQLTables() function returns a list of table names and associated information that is stored in the system catalog of the connected data source.

The list of table names is returned as a result set, which you can retrieve by using the same functions that you use to process a result set that is generated by a query.

### Purpose

Specification:	CLI 2.1	ODBC 1.0	
----------------	---------	----------	--

## SQLTables function (CLI) - Get table information

**Unicode equivalent:** You can also use this function with the Unicode character set. The corresponding Unicode function is `SQLTablesW()`. For information about ANSI and Unicode function mappings, see “Unicode functions (CLI)” on page 5.

### Syntax

```
SQLRETURN SQLTables (
    SQLHSTMT StatementHandle, /* hstmt */
    SQLCHAR *CatalogName, /* szCatalogName */
    SQLSMALLINT NameLength1, /* cbCatalogName */
    SQLCHAR *SchemaName, /* szSchemaName */
    SQLSMALLINT NameLength2, /* cbSchemaName */
    SQLCHAR *TableName, /* szTableName */
    SQLSMALLINT NameLength3, /* cbTableName */
    SQLCHAR *TableType, /* szTableType */
    SQLSMALLINT NameLength4); /* cbTableType */
```

### Function arguments

Table 156. SQLTables arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	The statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	A catalog qualifier of a 3-part table name that can contain a <i>pattern value</i> . If the target DBMS does not support 3-part naming, and <i>CatalogName</i> is not a null pointer and does not point to a zero-length string, then an empty result set and <code>SQL_SUCCESS</code> is returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	<i>NameLength1</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>CatalogName</i> , or <code>SQL_NTS</code> if <i>CatalogName</i> is null-terminated.
SQLCHAR *	<i>SchemaName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by the schema name.
SQLSMALLINT	<i>NameLength2</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>SchemaName</i> , or <code>SQL_NTS</code> if <i>SchemaName</i> is null-terminated.
SQLCHAR *	<i>TableName</i>	Input	A buffer that can contain a <i>pattern value</i> to qualify the result set by the table name.
SQLSMALLINT	<i>NameLength3</i>	Input	The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableName</i> , or <code>SQL_NTS</code> if <i>TableName</i> is null-terminated.



Table 156. SQLTables arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>TableType</i>	Input	<p>A buffer that can contain a <i>value list</i> to qualify the result set by table type.</p> <p>The value list is a list of uppercase comma-separated single values for the table types of interest. Valid table type identifiers include: ALIAS, HIERARCHY TABLE, INOPERATIVE VIEW, NICKNAME, MATERIALIZED QUERY TABLE, SYSTEM TABLE, TABLE, TYPED TABLE, TYPED VIEW, or VIEW. If <i>TableType</i> argument is a NULL pointer or a zero length string, this is equivalent to specifying all of the possibilities for the table type identifier.</p> <p>If SYSTEM TABLE is specified, both system tables and system views (if there are any) are returned.</p>
SQLSMALLINT	<i>NameLength4</i>	Input	<p>The number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) that are required to store <i>TableType</i>, or SQL_NTS if <i>TableType</i> is null-terminated.</p>

The *CatalogName*, *SchemaName*, and *TableName* input arguments accept search patterns.

## Usage

Table information is returned in a result set where each table is represented by one row of the result set. To determine the type of access that is permitted on any given table in the list, the application can call the `SQLTablePrivileges()` function. The application must be able to handle a situation where the user selects a table for which SELECT privileges are not granted.

To support obtaining just a list of schemas, you can apply the succeeding semantics for the *SchemaName* argument: if *SchemaName* is a string that contains a single percent (%) character, and *CatalogName* and *TableName* are empty strings, then the result set contains a list of valid schemas in the data source.

If *TableType* is a single percent character (%) and *CatalogName*, *SchemaName*, and *TableName* are empty strings, then the result set contains a list of valid table types for the data source. (All columns except the TABLE\_TYPE column contain NULLs.)

If *TableType* is not an empty string, it must contain a list of uppercase, comma-separated values for the types of interest. You can enclose each value in single quotation marks or place double quotation marks around all of the values. For example, "'TABLE','VIEW'" or "TABLE,VIEW". If the data source does not support or does not recognize a specified table type, nothing is returned for that type.

Sometimes, an application calls the `SQLTables()` function with null pointers for some or all of the *SchemaName*, *TableName*, and *TableType* arguments so that no attempt is made to restrict the result set that is returned. For some data sources that contain a large quantity of tables, views and, aliases for example, this scenario maps to an extremely large result set and very long retrieval times. You can specify three configuration keywords (SCHEMALIST, SYSSHEMA, TABLETYPE) in the CLI initialization file to help restrict the result set when the application has

## SQLTables function (CLI) - Get table information

supplied null pointers for *SchemaName*, *TableType*, or both. If the application specifies a *SchemaName* string, the SCHEMALIST keyword is still used to restrict the output. Therefore, if the schema name that is supplied is not in the SCHEMALIST string, the result is an empty result set.

The result set that is returned by the SQLTables() function contains the columns that are listed in Columns returned by SQLTables in the order given. The rows are ordered by the TABLE\_TYPE, TABLE\_CAT, TABLE\_SCHEM, and TABLE\_NAME columns.

In many cases, calls to the SQLTables() function map to a complex and thus expensive query against the system catalog, so you should use the calls sparingly, and save the results rather than repeating calls.

Call the SQLGetInfo() function with the SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_OWNER\_SCHEMA\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN to determine respectively the actual lengths of the TABLE\_CAT, TABLE\_SCHEM, TABLE\_NAME, and COLUMN\_NAME columns that are supported by the connected DBMS.

In Version 9.7 Fix Pack 5, you can specify \*ALL or \*USRLIBL as values in the *SchemaName* to resolve unqualified stored procedure calls or to find libraries in catalog API calls. If you specify \*ALL, CLI searches on all existing schemas in the connected database. You are not required to specify \*ALL, as this behavior is the default in CLI. For IBM DB2 for IBM i servers, if you specify \*USRLIBL, CLI searches on the current libraries of the server job. For other DB2 servers, \*USRLIBL does not have a special meaning and CLI searches using \*USRLIBL as a pattern. Alternatively, you can set the SchemaFilter IBM Data Server Driver configuration keyword or the Schema List CLI/ODBC configuration keyword to \*ALL or \*USRLIBL.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

### Columns returned by SQLTables

#### Column 1 TABLE\_CAT (VARCHAR(128))

Name of the catalog containing TABLE\_SCHEM. The value is NULL if this table does not have catalogs.

#### Column 2 TABLE\_SCHEM (VARCHAR(128))

Name of the schema containing TABLE\_NAME.

#### Column 3 TABLE\_NAME (VARCHAR(128))

Name of the table, view, alias, or synonym.

#### Column 4 TABLE\_TYPE (VARCHAR(128))

Identifies the type that is given by the name in the TABLE\_NAME column. It can have the string values 'ALIAS', 'HIERARCHY TABLE', 'INOPERATIVE VIEW', 'NICKNAME', 'MATERIALIZED QUERY TABLE', 'SYSTEM TABLE', 'TABLE', 'TYPED TABLE', 'TYPED VIEW', or 'VIEW'.

#### Column 5 REMARKS (VARCHAR(254))

Descriptive information about the table.

### Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## SQLTables function (CLI) - Get table information

- SQL\_STILL\_EXECUTING
- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

### Diagnostics

Table 157. SQLTables SQLSTATES

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 08S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for <i>StatementHandle</i> . The function was called and before it completed execution, <code>SQLCancel()</code> was called on <i>StatementHandle</i> from a different thread in a multithreaded application. Then the function was called again on <i>StatementHandle</i> .
HY009	Invalid argument value.	<i>TableName</i> is null.
HY010	Function sequence error.	<p>The function was called during a data-at-execute (<code>SQLParamData()</code>, <code>SQLPutData()</code>) operation.</p> <p>The function was called while in a BEGIN COMPOUND and END COMPOUND SQL operation.</p> <p>An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>The function was called before a statement was prepared on the statement handle.</p>
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	<p>The value of one of the name length arguments was less than 0, but not equal to <code>SQL_NTS</code>.</p> <p>The valid of one of the name length arguments exceeded the maximum value supported for that data source. The maximum supported value can be obtained by calling the <code>SQLGetInfo()</code> function.</p>
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the <code>SQL_ATTR_QUERY_TIMEOUT</code> attribute for <code>SQLSetStmtAttr()</code> .

### Restrictions

None.

### Example

```
/* get table information */
cliRC = SQLTables(hstmt,
                 NULL,
```

## SQLTables function (CLI) - Get table information

```
0,  
tbSchemaPattern,  
SQL_NTS,  
tbNamePattern,  
SQL_NTS,  
NULL,  
0);
```

---

## SQLTransact function (CLI) - Transaction management

### Deprecated

#### Note:

In ODBC 3.0, SQLTransact() has been deprecated and replaced with SQLEndTran().

Although this version of CLI continues to support SQLTransact(), use SQLEndTran() in your CLI programs so that they conform to the latest standards.

#### Migrating to the new function

The statement:

```
SQLTransact(henv, hdbc, SQL_COMMIT);
```

for example, would be rewritten using the new function as:

```
SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
```

---

## Chapter 2. Return codes and SQLSTATES for CLI

When you call CLI functions, there are two levels of diagnostics returned by the function: return codes and detailed diagnostics (SQLSTATES, messages, SQLCA).

Each CLI function returns the function return code as a basic diagnostic. Both `SQLGetDiagRec()` and `SQLGetDiagField()` provide more detailed diagnostic information. If the diagnostic originates at the DBMS, the `SQLGetSQLCA()` function provides access to the SQLCA. This arrangement lets applications handle the basic flow control based on return codes, and use the SQLSTATES along with the SQLCA to determine the specific causes of failure and to perform specific error handling.

Both `SQLGetDiagRec()` and `SQLGetDiagField()` return three pieces of information:

- SQLSTATE
- Native error: if the diagnostic is detected by the data source, this is the SQLCODE; otherwise, this is set to -99999.
- Message text: this is the message text associated with the SQLSTATE.

`SQLGetSQLCA()` returns the SQLCA for access to specific fields, but should only be used when `SQLGetDiagRec()` or `SQLGetDiagField()` cannot provide the required information.

---

### CLI function return codes

The following table lists all possible return codes for CLI functions.

*Table 158. CLI Function return codes*

Return code	Explanation
SQL_SUCCESS	The function completed successfully, no additional SQLSTATE information is available.
SQL_SUCCESS_WITH_INFO	The function completed successfully with a warning or other information. Call <code>SQLGetDiagRec()</code> or <code>SQLGetDiagField()</code> to receive the SQLSTATE and any other informational messages or warnings. The SQLSTATE will have a class of '01'.
SQL_STILL_EXECUTING	The function is running asynchronously and has not yet completed. The CLI driver has returned control to the application after calling the function, but the function has not yet finished executing.
SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. When this is returned after the execution of an SQL statement, additional information may be available and can be obtained by calling <code>SQLGetDiagRec()</code> or <code>SQLGetDiagField()</code> .
SQL_NEED_DATA	The application tried to execute an SQL statement but CLI lacks parameter data that the application had indicated would be passed at execute time.
SQL_ERROR	The function failed. Call <code>SQLGetDiagRec()</code> or <code>SQLGetDiagField()</code> to receive the SQLSTATE and any other error information.

## CLI function return codes

Table 158. CLI Function return codes (continued)

Return code	Explanation
SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle). This is a programming error. No further information is available.

The following code segment shows how a function return code, `SQL_NO_DATA_FOUND`, can be used to control when data retrieval should stop:

```
while (cliRC != SQL_NO_DATA_FOUND)
{
    printf("    %-8d %-14.14s \n", deptnumb.val, location.val);
    /* fetch next row */
    cliRC = SQLFetch(hstmt);
    STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
}
```

---

## SQLSTATES for CLI

SQLSTATES are alphanumeric strings of 5 characters (bytes) with a format of `ccsss`, where `cc` indicates class and `sss` indicates subclass. Any SQLSTATE that has a class of:

- '01', is a warning.
- 'HY', is generated by the CLI or ODBC driver.
- 'IM', is generated by the ODBC driver manager.

**Note:** Versions of CLI before Version 5 returned SQLSTATES with a class of 'S1' rather than 'HY'. To force the CLI driver to return 'S1' SQLSTATES, the application should set the environment attribute `SQL_ATTR_ODBC_VERSION` to the value `SQL_OV_ODBC2`.

CLI SQLSTATES include both additional IBM defined SQLSTATES that are returned by the database server, and CLI defined SQLSTATES for conditions that are not defined in the ODBC version 3 and ISO SQL/CLI specifications. This allows for the maximum amount of diagnostic information to be returned. When running applications in an ODBC environment, it is also possible to receive ODBC defined SQLSTATES.

Follow these guidelines for using SQLSTATES within your application:

- Always check the function return code before calling `SQLGetDiagRec()` to determine if diagnostic information is available.
- Use the SQLSTATES rather than the native error code.
- To increase your application's portability, only build dependencies on the subset of CLI SQLSTATES that are defined by the ODBC version 3 and ISO SQL/CLI specifications, and return the additional ones as information only. A dependency in an application is a logic flow decision based on specific SQLSTATES.

**Note:** It may be useful to build dependencies on the class (the first 2 characters) of the SQLSTATES.

- For maximum diagnostic information, return the text message along with the SQLSTATE (if applicable, the text message will also include the IBM defined SQLSTATE). It is also useful for the application to print out the name of the function that returned the error.

- Ensure that the string allocated for the SQLSTATE includes space for the null termination character returned by CLI.

The following code segment from `utilcli.c` shows how diagnostic information, such as SQLSTATES, can be retrieved and displayed:

```
void HandleDiagnosticsPrint(SQLSMALLINT htype, /* handle type identifier */
                           SQLHANDLE hndl /* handle */)
{
    SQLCHAR message[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
    SQLINTEGER sqlcode;
    SQLSMALLINT length, i;

    i = 1;

    /* get multiple field settings of diagnostic record */
    while (SQLGetDiagRec(htype,
                        hndl,
                        i,
                        sqlstate,
                        &sqlcode,
                        message,
                        SQL_MAX_MESSAGE_LENGTH + 1,
                        &length) == SQL_SUCCESS)
    {
        printf("\n SQLSTATE          =
Native Error Code =
");
        printf("
");
        i++;
    }

    printf("-----\n");
}
```

You can use the CLI/ODBC trace facility to gain a better understanding of how your application calls DB2, including any errors that may occur.

---

## Return codes for compound SQL (CLI) in CLI applications

Return codes are generated on the call to `SQLExecute()` or `SQLExecDirect()` for the END COMPOUND statement. The following lists the return codes for ATOMIC and NOT ATOMIC compound statements:

### ATOMIC

- `SQL_SUCCESS`: all substatements have executed without any warnings or errors.
- `SQL_SUCCESS_WITH_INFO`: all substatements executed successfully with one or more warnings. Call `SQLGetDiagRec()` or `SQLGetDiagField()` to retrieve additional information on the error or warning. The handle used by `SQLGetDiagRec()` or `SQLGetDiagField()` must be the same one used to process the BEGIN COMPOUND and END COMPOUND statements.
- `SQL_NO_DATA_FOUND`: BEGIN COMPOUND and END COMPOUND statements executed without any substatements, or none of the substatements affected any rows.
- `SQL_ERROR`: one or more substatements failed and all substatements were rolled back.

### NOT ATOMIC

- `SQL_SUCCESS`: all substatements executed without any errors.

## Return codes for compound SQL (CLI) in CLI applications

- `SQL_SUCCESS_WITH_INFO`: the `COMPOUND` statement executed with one or more warnings returned by one or more substatements. Call `SQLGetDiagRec()` or `SQLGetDiagField()` to retrieve additional information on the error or warning. The handle used by `SQLGetDiagRec()` or `SQLGetDiagField()` must be the same one used to process the `BEGIN COMPOUND` and `END COMPOUND` statements.
- `SQL_NO_DATA_FOUND`: the `BEGIN COMPOUND` and `END COMPOUND` statements executed without any substatements, or none of the substatements affected any rows.
- `SQL_ERROR`: the `COMPOUND` statement failed. At least one substatement returned an error. Examine the `SQLCA` to determine which statements failed.



---

## Chapter 3. CLI/ODBC configuration keywords listing by category

Categorized list of CLI configuration keywords

The CLI/ODBC configuration keywords are divided into the listed categories:

- Compatibility configuration keywords
- Data source configuration keywords
- Data type configuration keywords
- Enterprise configuration keywords
- Environment configuration keywords
- File DSN configuration keywords
- Optimization configuration keywords
- Service configuration keywords
- Static SQL configuration keywords
- Transaction configuration keywords

While most CLI/ODBC configuration keywords can be set in the `db2cli.ini` initialization file or by providing the keyword information in the connection string to `SQLDriverConnect()` function, the **Trusted\_Connection** Keyword can be set only with `SQLDriverConnect()`.

### Compatibility configuration keywords

Use the compatibility configuration keywords to define DB2 behavior. You can set the compatibility configuration keywords to ensure that other applications are compatible with DB2.

- “AllowInterleavedGetData CLI/ODBC configuration keyword” on page 325
- “CheckForFork CLI/ODBC configuration keyword” on page 335
- “CursorTypes CLI/ODBC configuration keyword” on page 348
- “DeferredPrepare CLI/ODBC configuration keyword” on page 354
- “DescribeCall CLI/ODBC configuration keyword” on page 355
- “DescribeParam CLI/ODBC configuration keyword” on page 357
- “DisableKeysetCursor CLI/ODBC configuration keyword” on page 358
- “DisableMultiThread CLI/ODBC configuration keyword” on page 358
- “DisableUnicode CLI/ODBC configuration keyword” on page 359
- “EnableNamedParameterSupport CLI/ODBC configuration keyword” on page 359
- “Interrupt CLI/ODBC configuration keyword” on page 365
- “OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword” on page 377
- “OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword” on page 378
- “RetCatalogAsCurrServer CLI/ODBC configuration keyword” on page 392
- “RetOleDbConnStr CLI/ODBC configuration keyword” on page 392

## CLI/ODBC configuration keywords listing by category

- “Trusted\_Connection CLI/ODBC configuration keyword” on page 420 (use `SQLDriverConnect()` function to set this keyword.)
- “TimestampTruncErrToWarning CLI/ODBC configuration keyword” on page 407

### Data source configuration keywords

General keywords associated with a data source configuration.

- “DBAlias CLI/ODBC configuration keyword” on page 351
- “ClientEncAlg CLI/ODBC configuration keyword” on page 337
- “PWD CLI/ODBC configuration keyword” on page 379
- “UID CLI/ODBC configuration keyword” on page 422

### Data type configuration keywords

Use the data type configuration keywords to define how DB2 reports and handles various data types.

- “BitData CLI/ODBC configuration keyword” on page 333
- “CurrentImplicitXMLParseOption CLI/ODBC configuration keyword” on page 344
- “DateTimeStringFormat CLI/ODBC configuration keyword” on page 352
- “DecimalFloatRoundingMode CLI/ODBC configuration keyword” on page 353
- “FloatPrecRadix CLI/ODBC configuration keyword” on page 360
- “Graphic CLI/ODBC configuration keyword” on page 363
- “LOBMaxColumnSize CLI/ODBC configuration keyword” on page 367
- “LongDataCompat CLI/ODBC configuration keyword” on page 368
- “MapBigintCDefault CLI/ODBC configuration keyword” on page 369
- “MapCharToWChar CLI/ODBC configuration keyword” on page 369
- “MapDateCDefault CLI/ODBC configuration keyword” on page 370
- “MapDateDescribe CLI/ODBC configuration keyword” on page 370
- “MapDecimalFloatDescribe CLI/ODBC configuration keyword” on page 371
- “MapGraphicDescribe CLI/ODBC configuration keyword” on page 372
- “MapTimeCDefault CLI/ODBC configuration keyword” on page 372
- “MapTimeDescribe CLI/ODBC configuration keyword” on page 373
- “MapTimestampCDefault CLI/ODBC configuration keyword” on page 373
- “MapTimestampDescribe CLI/ODBC configuration keyword” on page 374
- “MapXMLCDefault CLI/ODBC configuration keyword” on page 375
- “MapXMLDescribe CLI/ODBC configuration keyword” on page 375
- “OleDbReturnCharAsWChar CLI/ODBC configuration keyword” on page 377
- “PromoteLONGVARtoLOB CLI/ODBC configuration keyword” on page 388
- “XMLDeclaration CLI/ODBC configuration keyword” on page 425

### Enterprise configuration keywords

Use the enterprise configuration keywords to maximize the efficiency of connections to large databases.

- “ConnectNode CLI/ODBC configuration keyword” on page 341
- “CurrentPackagePath CLI/ODBC configuration keyword” on page 345
- “CurrentPackageSet CLI/ODBC configuration keyword” on page 346

## CLI/ODBC configuration keywords listing by category

- “CurrentRefreshAge CLI/ODBC configuration keyword” on page 346
- “CurrentSchema CLI/ODBC configuration keyword” on page 347
- “CurrentSQLID CLI/ODBC configuration keyword” on page 347
- “DBName CLI/ODBC configuration keyword” on page 351
- “GranteeList CLI/ODBC configuration keyword” on page 362
- “GrantorList CLI/ODBC configuration keyword” on page 362
- “OnlyUseBigPackages CLI/ODBC configuration keyword” on page 379
- “ReportPublicPrivileges CLI/ODBC configuration keyword” on page 391
- “ReturnSynonymSchema CLI/ODBC configuration keyword” on page 394
- “SchemaList CLI/ODBC configuration keyword” on page 396
- “ServerMsgMask CLI/ODBC configuration keyword” on page 397
- “SQLCODEMAP CLI/ODBC configuration keyword” on page 399
- “SysSchema CLI/ODBC Configuration Keyword” on page 404
- “TableType CLI/ODBC configuration keyword” on page 405
- “UseServerMsgSP CLI/ODBC configuration keyword” on page 423

### Environment configuration keywords

Use the environment configuration keywords to define environment-specific settings, such as the location of various files on the server and client machines.

- 
- “ConnectTimeout CLI/ODBC configuration keyword” on page 342
- “CurrentFunctionPath CLI/ODBC configuration keyword” on page 343
- “Interrupt CLI/ODBC configuration keyword” on page 365
- “QueryTimeoutInterval CLI/ODBC configuration keyword” on page 389
- “ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword” on page 390
- “ReceiveTimeout CLI/ODBC configuration keyword” on page 390
- “TempDir CLI/ODBC configuration keyword” on page 406

### File DSN configuration keywords

Use the File DSN configuration keywords to set the TCP/IP settings for a file DSN connection.

- “Attach CLI/ODBC configuration keyword” on page 330
- “Authentication CLI/ODBC configuration keyword” on page 331
- “BIDI CLI/ODBC configuration keyword” on page 333
- “Database CLI/ODBC configuration keyword” on page 352
- “Hostname CLI/ODBC configuration keyword” on page 363
- “Port CLI/ODBC configuration keyword” on page 386
- “Protocol CLI/ODBC configuration keyword” on page 388
- “ServiceName CLI/ODBC configuration keyword” on page 398
- “security CLI/ODBC configuration keyword” on page 397
- “SSLClientLabel CLI/ODBC configuration keyword” on page 399
- “SSLClientKeystoredb CLI/ODBC configuration keyword” on page 400
- “SSLClientKeystoreDBPassword CLI/ODBC configuration keyword” on page 401

## CLI/ODBC configuration keywords listing by category

- “SSLClientKeystash CLI/ODBC configuration keyword” on page 400
- “TargetPrincipal CLI/ODBC configuration keyword” on page 406

### Optimization configuration keywords

Use the optimization configuration keywords to speed up and reduce the amount of network flow between the IBM Data Server Driver for ODBC and CLI and the server.

- “AllowGetDataLOBReaccess CLI/ODBC configuration keyword” on page 325
- “AppendForFetchOnly CLI/ODBC configuration keyword” on page 328
- “AppUsesLOBLocator CLI/ODBC configuration keyword” on page 327
- “BlockForNRows CLI/ODBC configuration keyword” on page 333
- “BlockLobs CLI/ODBC configuration keyword” on page 334
- “ClientBuffersUnboundLOBS CLI/ODBC configuration keyword” on page 337
- “ColumnwiseMRI CLI/ODBC configuration keyword” on page 339
- “ConcurrentAccessResolution CLI/ODBC configuration keyword” on page 340
- “CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword” on page 344
- “DB2Degree CLI/ODBC configuration keyword” on page 348
- “DB2Explain CLI/ODBC configuration keyword” on page 349
- “DB2NETNamedParam CLI/ODBC configuration keyword” on page 350
- “DB2Optimization CLI/ODBC configuration keyword” on page 350
- “DescribeInputOnPrepare CLI/ODBC configuration keyword” on page 355
- “DescribeOutputLevel CLI/ODBC configuration keyword” on page 356
- “FET\_BUF\_SIZE CLI/ODBC configuration keyword” on page 360
- “GetDataLobNoTotal CLI/ODBC configuration keyword” on page 361
- “KeepDynamic CLI/ODBC configuration keyword” on page 366
- “LOBCacheSize CLI/ODBC configuration keyword” on page 366
- “LOBFileThreshold CLI/ODBC configuration keyword” on page 367
- “LockTimeout CLI/ODBC configuration keyword” on page 368
- “MaxLOBBlockSize CLI/ODBC configuration keyword” on page 376
- “OptimizeForNRows CLI/ODBC configuration keyword” on page 379
- “Reopt CLI/ODBC configuration keyword” on page 390
- “ReturnAliases CLI/ODBC configuration keyword” on page 394
- “SkipTrace CLI/ODBC configuration keyword” on page 398
- “StmtConcentrator CLI/ODBC configuration keyword” on page 403
- “StreamGetData CLI/ODBC configuration keyword” on page 403
- “StreamPutData CLI/ODBC configuration keyword” on page 404
- “Underscore CLI/ODBC configuration keyword” on page 422

### Service configuration keywords

Use the service configuration keywords to help in troubleshooting problems with CLI/ODBC connections. Programmers can also use service configuration keywords to gain a better understanding of how their CLI programs are translated into calls to the server.

- “AppendAPIName CLI/ODBC configuration keyword” on page 327

## CLI/ODBC configuration keywords listing by category

- “AppendRowColToErrorMessage CLI/ODBC configuration keyword” on page 328
- “IgnoreWarnings CLI/ODBC configuration keyword” on page 364
- “IgnoreWarnList CLI/ODBC configuration keyword” on page 364
- “LoadXAInterceptor CLI/ODBC configuration keyword” on page 368
- “Patch1 CLI/ODBC configuration keyword” on page 380
- “Patch2 CLI/ODBC configuration keyword” on page 383
- “ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword” on page 391
- “RetryOnError CLI/ODBC configuration keyword” on page 393
- “ProgramID CLI/ODBC configuration keyword” on page 387
- “ProgramName CLI/ODBC configuration keyword” on page 387
- “Trace CLI/ODBC configuration keyword” on page 407
- “TraceAPIList CLI/ODBC configuration keyword” on page 408
- “TraceAPIList! CLI/ODBC configuration keyword” on page 410
- “TraceComm CLI/ODBC configuration keyword” on page 412
- “TraceErrImmediate CLI/ODBC configuration keyword” on page 413
- “TraceFileName CLI/ODBC configuration keyword” on page 413
- “TraceFlush CLI/ODBC configuration keyword” on page 414
- “TraceFlushOnError CLI/ODBC configuration keyword” on page 415
- “TraceLocks CLI/ODBC configuration keyword” on page 415
- “TracePathName CLI/ODBC configuration keyword” on page 417
- “TracePIDList CLI/ODBC configuration keyword” on page 416
- “TracePIDTID CLI/ODBC configuration keyword” on page 416
- “TraceRefreshInterval CLI/ODBC configuration keyword” on page 418
- “TraceStmtOnly CLI/ODBC configuration keyword” on page 418
- “TraceTime CLI/ODBC configuration keyword” on page 419
- “TraceTimestamp CLI/ODBC configuration keyword” on page 419
- “WarningList CLI/ODBC configuration keyword” on page 424

### Static SQL configuration keywords

Use the static SQL configuration keywords when running static SQL statements in CLI/ODBC applications.

- “StaticCapFile CLI/ODBC configuration keyword” on page 401
- “StaticLogFile CLI/ODBC configuration keyword” on page 402
- “StaticMode CLI/ODBC configuration keyword” on page 402
- “StaticPackage CLI/ODBC configuration keyword” on page 402

### Transaction configuration keywords

Use the transaction configuration keywords to control and speed up SQL statements that are used in the application.

- “ArrayInputChain CLI/ODBC configuration keyword” on page 329
- “AsyncEnable CLI/ODBC configuration keyword” on page 330
- “AutoCommit CLI/ODBC configuration keyword” on page 332
- “ClientAcctStr CLI/ODBC configuration keyword” on page 336
- “ClientApplName CLI/ODBC configuration keyword” on page 336

## CLI/ODBC configuration keywords listing by category

- “ClientUserID CLI/ODBC configuration keyword” on page 338
- “ClientWrkStnName CLI/ODBC configuration keyword” on page 339
- “CommitOnEOF CLI/ODBC configuration keyword” on page 340
- “ConnectType CLI/ODBC configuration keyword” on page 343
- “CursorHold CLI/ODBC configuration keyword” on page 347
- “Mode CLI/ODBC configuration keyword” on page 376
- “SQLOverrideFileName CLI/ODBC configuration keyword” on page 395
- “TxnIsolation CLI/ODBC configuration keyword” on page 421
- “UseOldStpCall CLI/ODBC configuration keyword” on page 423

---

## db2cli.ini initialization file

The CLI/ODBC initialization file (`db2cli.ini`) contains various keywords and values that can be used to configure the behavior of CLI and the applications using it.

The keywords are associated with the database alias name, and affect all CLI and ODBC applications that access the database.

In Version 9.7 Fix Pack 3 and later, the `db2cli.ini.sample` sample configuration file is shipped to help you get started. You can create a `db2cli.ini` file that is based on the `db2cli.ini.sample` file and that is stored in the same location. The location of the sample configuration file depends on your driver type and platform.

For IBM Data Server Client, IBM Data Server Runtime Client, or IBM Data Server Driver Package, the sample configuration file is created in one of the following paths:

- On AIX®, HP-UX, Linux, or Solaris operating systems: `instance_path/cfg`
- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\DB2\driver_copy_name\cfg`
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\DB2\driver_copy_name\cfg`

For example, if you use IBM Data Server Driver Package for Windows XP, and the data server driver copy name is `DSD_COPY`, then the `db2cli.ini.sample` file is created in the `C:\Documents and Settings\All Users\Application Data\IBM\DB2\DSD_COPY\cfg` directory.

For IBM Data Server Driver for ODBC and CLI, the sample configuration file is created in one of the following paths:

- On AIX, HP-UX, Linux, or Solaris operating systems: `instance_path/cfg`
- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\DB2\driver_installation_path\cfg`  
where `driver_installation_path` is the file path where the driver is installed, with each directory separated by an underscore (`_`) instead of a backslash (`\`).
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\DB2\driver_installation_path\cfg`  
where `driver_installation_path` is the file path where the driver is installed, with each directory separated by an underscore (`_`) instead of a backslash (`\`).

For example, if you use IBM Data Server Driver for ODBC and CLI for Windows Vista, and the driver is installed in the `C:\IBMDB2\CLIDRIVER\V97FP3` directory, then

the `db2cli.ini.sample` file is created in the `C:\ProgramData\IBM\DB2\C_IBMDB2_CLIDRIVER_V97FP3\cfg` directory.

When the ODBC Driver Manager is used to configure a user DSN on Windows operating systems, the `db2cli.ini` file is created in `Documents and Settings\User Name` where *User Name* represents the name of the user directory.

You can use the environment variable **DB2CLIINIPATH** to specify a different location for the `db2cli.ini` file.

If you use a copy of the `db2cli.ini` file from Version 9.7 Fix Pack 2 or earlier on Windows, the file is in a different location. You can keep the copy of the `db2cli.ini` file in the previous location, but this location might not be valid in future releases.

The configuration keywords enable you to:

- Configure general features such as data source name, user name, and password.
- Set options that will affect performance.
- Indicate query parameters such as wild card characters.
- Set patches or work-arounds for various ODBC applications.
- Set other, more specific features associated with the connection, such as code pages and IBM GRAPHIC data types.
- Override default connection options specified by an application. For example, if an application requests Unicode support from the CLI driver by setting the `SQL_ATTR_ANSI_APP` connection attribute, then setting **DisableUnicode=1** in the `db2cli.ini` file will force the CLI driver not to provide the application with Unicode support.

**Note:** If the CLI/ODBC configuration keywords set in the `db2cli.ini` file conflict with keywords in the `SQLDriverConnect()` connection string, then the `SQLDriverConnect()` keywords will take precedence.

The `db2cli.ini` initialization file is an ASCII file which stores values for the CLI configuration options. The `db2cli.ini.sample` sample configuration file is shipped to help you get started. While most CLI/ODBC configuration keywords are set in the `db2cli.ini` initialization file, some keywords are set by providing the keyword information in the connection string to `SQLDriverConnect()` instead.

There is one section within the file for each database (data source) the user wishes to configure. If needed, there is also a common section that affects all database connections.

Only the keywords that apply to all database connections through the CLI/ODBC driver are included in the **COMMON** section. This includes the following keywords:

- **CheckForFork**
- **DiagPath**
- **DisableMultiThread**
- **JDBCTrace**
- **JDBCTraceFlush**
- **JDBCTracePathName**
- **QueryTimeoutInterval**

## db2cli.ini initialization file

- **ReadCommonSectionOnNullConnect**
- **Trace**
- **TraceComm**
- **TraceErrImmediate**
- **TraceFileName**
- **TraceFlush**
- **TraceFlushOnError**
- **TraceLocks**
- **TracePathName**
- **TracePIDList**
- **TracePIDTID**
- **TraceRefreshInterval**
- **TraceStmtOnly**
- **TraceTime**
- **TraceTimeStamp**

All other keywords are to be placed in the database specific section, described below.

**Note:** Configuration keywords are valid in the COMMON section, however, they will apply to all database connections.

The COMMON section of the db2cli.ini file begins with:

```
[COMMON]
```

Before setting a common keyword it is important to evaluate its impact on all CLI/ODBC connections from that client. A keyword such as **TRACE**, for instance, will generate information on all CLI/ODBC applications connecting to DB2 on that client, even if you are intending to troubleshoot only one of those applications.

Each database specific section always begins with the name of the data source name (DSN) between square brackets:

```
[data source name]
```

This is called the *section header*.

The parameters are set by specifying a keyword with its associated keyword value in the form:

**KeywordName** =*keywordValue*

- All the keywords and their associated values for each database must be located below the database section header.
- If the database-specific section does not contain a **DBAlias** keyword, the data source name is used as the database alias when the connection is established. The keyword settings in each section apply only to the applicable database alias.
- The keywords are not case sensitive; however, their values can be if the values are character based.
- If a database is not found in the .INI file, the default values for these keywords are in effect.
- Comment lines are introduced by having a semicolon in the first position of a new line.
- Blank lines are permitted.



- If duplicate entries for a keyword exist, the first entry is used (and no warning is given).

The following is a sample .INI file with two database alias sections:

```
; This is a comment line.
[MYDB22]
AutoCommit=0
TableType="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CurrentSQLID=SAIID
TableType="'TABLE'"
SchemaList="'USER1',CURRENT SQLID,'USER2'"
```

Although you can edit the db2cli.ini file manually on all platforms, it is recommended that you use the Configuration Assistant if it is available on your platform or the **UPDATE CLI CONFIGURATION** command. You must add a blank line after the last entry if you manually edit the db2cli.ini file.

---

## AllowGetDataLOBReaccess CLI/ODBC configuration keyword

Specifies whether the application can call SQLGetData() for previously accessed LOB columns when querying database servers that support Dynamic Data Format.

### db2cli.ini keyword syntax:

```
AllowGetDataLOBReaccess = 0 | 1
```

### Default setting:

Do not allow calls to SQLGetData() for previously accessed LOB columns when querying database servers that support Dynamic Data Format.

### Usage notes:

This keyword only affects connections to database servers that support Dynamic Data Format, also known as progressive streaming. The default setting of 0 does not allow applications to call SQLGetData() for previously accessed LOB columns. Specify 1 to allow applications to call SQLGetData() for previously accessed LOB columns.

Note that when the keyword is set to 1 to allow re-access to LOB columns, some resources on the server might not be freed upon completion of SQLGetData().

If the server does not support Dynamic Data Format, this keyword has no effect and calls to SQLGetData() for previously accessed LOB columns are allowed.

A similar keyword exists called AllowInterleavedGetData that allows applications to call SQLGetData() for previously accessed LOB columns and maintain the data offset position from the previous call to SQLGetData() when querying data servers that support Dynamic Data Format. If both AllowGetDataLOBReaccess and AllowInterleavedGetData are set for a given connection or statement, the AllowInterleavedGetData setting takes precedence over AllowGetDataLOBReaccess.

---

## AllowInterleavedGetData CLI/ODBC configuration keyword

Specifies whether the application can call SQLGetData() for previously accessed LOB columns and maintain the data offset position from the previous call to SQLGetData() when querying data servers that support Dynamic Data Format.

## AllowInterleavedGetData CLI/ODBC configuration keyword

### db2cli.ini keyword syntax:

AllowInterleavedGetData = 0 | 1

### Default setting:

Do not allow calls to SQLGetData() for previously accessed LOB columns when querying database servers that support Dynamic Data Format.

### Equivalent environment or connection attribute:

SQL\_ATTR\_ALLOW\_INTERLEAVED\_GETDATA

### Usage notes:

This keyword affects only connections to database servers that support Dynamic Data Format, also known as progressive streaming. The default setting of 0 does not allow applications to call SQLGetData() for previously accessed LOB columns. Specify 1 to allow applications to call SQLGetData() for previously accessed LOB columns and start reading LOB data from where the application stopped reading during the previous read.

Note that when the keyword is set to 1 to allow re-access to LOB columns, some resources on the server might not be freed upon completion of SQLGetData().

If the server does not support Dynamic Data Format, this keyword has no effect, and calls to SQLGetData() for previously accessed LOB columns are allowed.

A similar keyword exists called AllowGetDataLOBReaccess that allows applications to call SQLGetData() for previously accessed LOB columns. However, if the AllowGetDataLOBReaccess keyword is used, data position and offset information is not maintained. When the LOB column is re-accessed after interleaving, SQLGetData() starts reading data from the beginning for that LOB data column. If both AllowGetDataLOBReaccess and AllowInterleavedGetData are set for a given connection or statement, the AllowInterleavedGetData setting takes precedence over AllowGetDataLOBReaccess.

---

## AltHostName CLI/ODBC configuration keyword

Specifies the alternate host name to be used if the primary server specified by HOSTNAME cannot be contacted (Client Reroute.)

### db2cli.ini keyword syntax:

AltHostName = *fully qualified alternate host name* | *IP address of node*

### Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies a fully qualified host name or the IP address of the node where the alternate server for the database resides.

If the primary server returns alternate server information, it will override this AltHostName setting. However, this keyword is read only. That means the db2cli.ini will not be updated with the alternate server information received from the primary server.

---

## AltPort CLI/ODBC configuration keyword

Specifies the alternate port to be used if the primary server specified by HOSTNAME and PORT cannot be contacted (Client Reroute.)

### db2cli.ini keyword syntax:

*AltPort = port number*

### Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies the port number of the alternate server of the database manager instance where the alternate server for the database resides.

If the primary server returns alternate server information, it will override this AltPort setting. However, this keyword is read only. That means the db2cli.ini will not be updated with the alternate server information received from the primary server.

---

## AppUsesLOBLocator CLI/ODBC configuration keyword

Specifies whether applications use LOB locators.

### db2cli.ini keyword syntax:

*AppUsesLOBLocator = 0 | 1*

### Default setting:

Applications are using LOB locators.

### Equivalent connection or statement attribute:

SQL\_ATTR\_APP\_USES\_LOB\_LOCATOR

### Usage notes:

The default setting of 1 indicates that applications are using LOB locators. For applications that do not use LOB locators and are querying data on a server that supports Dynamic Data Format, also known as progressive streaming, specify 0 to indicate that LOB locators are not used and allow the return of LOB data to be optimized.

This keyword is ignored for stored procedure result sets.

If the keyword is set to 0 and an application binds a LOB locator to a result set using SQLBindCol(), an Invalid conversion error will be returned by the SQLFetch() function.

---

## AppendAPIName CLI/ODBC configuration keyword

Appends the CLI/ODBC function name which generated an error to the error message text.

### db2cli.ini keyword syntax:

*AppendAPIName = 0 | 1*

### Default setting:

Do NOT display CLI function name.

### Usage notes:

The CLI function (API) name that generated an error is appended to the error message retrieved using SQLGetDiagRec() or SQLError(). The function name is enclosed in curly braces { }.

For example,

## AppendAPIName CLI/ODBC configuration keyword

```
[IBM][CLI Driver]" CLIxxxx: < text >  
SQLSTATE=XXXXX {SQLGetData}"
```

- 0 = do NOT append CLI function name (default)
- 1 = append the CLI function name

This keyword is only useful for debugging.

---

## AppendForFetchOnly CLI/ODBC configuration keyword

Specifies whether the clause FOR FETCH ONLY is appended to READ-ONLY SQL statements.

**db2cli.ini keyword syntax:**

```
AppendForFetchOnly = 0 | 1
```

**Default setting:**

The keyword is not set by default. CLI appends the "FOR FETCH ONLY" clause only when connected to certain server types.

**Equivalent connection attribute:**

```
SQL_ATTR_APPEND_FOR_FETCH_ONLY
```

**Usage notes:**

By default, CLI appends the "FOR FETCH ONLY" clause to read SELECT statements when connected to DB2 for z/OS or DB2 for i databases.

This keyword allows an application to control when CLI appends the "FOR FETCH ONLY" clause, for example, in a situation where an application is binding the CLI packages using different bind BLOCKING options (for example, BLOCKING UNAMBIG) and wants to suppress the blocking in order to keep positioned on a given row.

To change the default CLI behavior, the keyword can be set as follows:

- 0: CLI never appends the "FOR FETCH ONLY" clause to read SELECT statements regardless of the server type it is connecting to.
- 1: CLI always appends the "FOR FETCH ONLY" clause to read SELECT statements regardless of the server type it is connecting to.

---

## AppendRowColToErrorMessage CLI/ODBC configuration keyword

Specifies whether the row and column numbers that generated the error are appended the error message string.

**db2cli.ini keyword syntax:**

```
AppendRowColToErrorMessage= 0 | 1
```

**Default setting:**

The default setting of 0 will return the error message string without the row and column numbers.

**Usage notes:**

Specify 1 to append the row and column number that generated the error to the error message string. The values for row and column numbers are only appended when DB2 CLI is able to apply a row or column number to the problem.

The row or column numbers appended to error messages are the same positive values that would be returned if an application called `SQLGetDiagField()` with the `DiagIdentifier` argument as `SQL_DIAG_ROW_NUMBER` or `SQL_DIAG_COLUMN_NUMBER`. When

## AppendRowColToErrorMessage CLI/ODBC configuration keyword

AppendRowColToErrorMessage is set to 1, errors returned from calls to SQLGetDescField(), SQLGetDescRec() or SQLError() will have these row or column numbers appended with the following format: Row=<r>, Col=<c>, if they can be determined.

For example, the default text for error CLI0111E is as follows:

```
[IBM][CLI Driver] CLI0111E Numeric value out of range. SQLSTATE=22003
```

Specifying 1 to append the row and column number will return the following text for error CLI0111E:

```
[IBM][CLI Driver] CLI0111E Numeric value out of range.  
SQLSTATE=22003 {Row=2,Col=1}
```

**Note:** It is also possible for an error to be returned with only a row number.

---

## ArrayInputChain CLI/ODBC configuration keyword

Enables array input without needing pre-specified size and memory allocation requirements of normal array input.

### db2cli.ini keyword syntax:

```
ArrayInputChain = -1 | 0 | <positive integer>
```

### Default setting:

Normal input array is enabled, where the array and its size must be specified before the corresponding SQLExecute() call is made.

### Usage notes:

By default, array input (where an array of values is bound to an input parameter) requires the array and its size to be specified before the corresponding SQLExecute() function is called. An application, however, may not know the array size in advance, or the array size may be too large for the application to allocate from its pool of available memory. Under these circumstances, the application can set ArrayInputChain=-1 and use the SQL\_ATTR\_CHAINING\_BEGIN and SQL\_ATTR\_CHAINING\_END statement attributes to enable chaining, which allows array input without the pre-specified size and memory requirements of normal array input.

To enable chaining:

1. Set the keyword ArrayInputChain = -1.
2. Prepare and bind input parameters to the SQL statement.
3. Set the SQL\_ATTR\_CHAINING\_BEGIN statement attribute with SQLSetStmtAttr().
4. Update the bound parameters with input data and call SQLExecute().
5. Repeat Step 4 for as many rows as there are in the input array.
6. Set the SQL\_ATTR\_CHAINING\_END statement attribute with SQLSetStmtAttr() after the last row in the array has been processed according to Step 4.

The effect of completing these steps will be the same as if normal array input had been used.

## ArrayInputChain CLI/ODBC configuration keyword

Setting ArrayInputChain=0 (the default value) turns this array input feature off. ArrayInputChain can also be set to any positive integer which sets the array size to use for the input array.

**Restriction:** DB2 CLI does not support array input chaining for compound SQL (compiled) or compound SQL (inlined) statements.

---

## AsyncEnable CLI/ODBC configuration keyword

Enables or disables the ability to execute queries asynchronously.

**db2cli.ini keyword syntax:**

AsyncEnable = 0 | 1

**Default setting:**

Queries can be executed asynchronously.

**Usage notes:**

This option allows you to enable or disable support that allows queries to execute asynchronously. This only benefits applications that were written to take advantage of this feature by setting the SQL\_ATTR\_ASYNC\_ENABLE attribute using SQLSetStmtAttr() or SQLSetConnectAttr().

- 0 = Queries are not executed asynchronously
- 1 = Allow queries to be executed asynchronously. The application must also enable the asynchronous functionality by setting SQL\_ATTR\_ASYNC\_ENABLE using SQLSetStmtAttr() or SQLSetConnectAttr(). (default)

Once a function has been called asynchronously, only the original function, SQLAllocHandle(), SQLCancel(), SQLSetStmtAttr(), SQLGetDiagField(), SQLGetDiagRec(), or SQLGetFunctions() can be called on the statement handle, until the original function returns a code other than SQL\_STILL\_EXECUTING. Any other function called on any other statement handle under the same connection returns SQL\_ERROR with an SQLSTATE of HY010 (Function sequence error).

---

## Attach CLI/ODBC configuration keyword

Specifies whether to attach to the server instance. You can specify this keyword in the connection string or set it in the db2cli.ini or db2dsdriver.cfg file.

**db2cli.ini keyword syntax:**

ATTACH = TRUE | FALSE

**Default setting:**

The SQLDriverConnect() function connects to the specified database.

**Equivalent environment or connection attribute:**

N/A

**Usage notes:**

When you set the keyword to TRUE, the SQLDriverConnect() function does not connect to a database but instead connects to the specified server instance.

## Attach CLI/ODBC configuration keyword

To establish a connection with a DB2 server instance for Linux, UNIX, and Windows remote server, the CLI application must specify values for the Hostname, Port, UID, PWD, and Protocol along with setting the keyword to TRUE.

Any value other than TRUE that you assign to the **ATTACH** keyword is treated as FALSE.

### Examples:

The following example shows a specification of the keyword in the `db2cli.ini` file:

```
ATTACH=TRUE
```

The following example shows specifications of the keyword in the `db2dsdriver.cfg` file:

```
<configuration>
  <dsncollection>
    <dsn
alias="db2dsn01",name="db2db01",host="server1.mynet.com",
port="50001">
      <parameter name="ATTACH" value="TRUE"/>
    </dsn>
  </dsncollection>
  <databases>
    <database name="sample", host="serv1.mynet.com",
port="50001">
      <parameter name="ATTACH" value="TRUE"/>
    </database>
  </databases>
</configuration>
```

## Version information

### Last update

This topic was last updated for IBM DB2 Version 9.7, Fix Pack 3.

### IBM Data Server Client

Supported in IBM DB2 Database for Linux, UNIX, and Windows

---

## Authentication CLI/ODBC configuration keyword

Specifies the type of authentication to be used with file DSN or DSN-less connectivity.

### db2cli.ini keyword syntax:

```
Authentication = CERTIFICATE | SERVER | SERVER_ENCRYPT |  
SERVER_ENCRYPT_AES | DATA_ENCRYPT | KERBEROS |  
GSSPLUGIN
```

### Default setting:

Not specified

### Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file, or in the connection string.

When you set this option, you must also set the following options:

- **Database**
- **Protocol**.

If **Protocol**=IPC, you need to set the following too:

- **Instance**.

## Authentication CLI/ODBC configuration keyword

If **Protocol**=TCPIP, you need to set the following too:

- **Port**
- **Hostname.**

If Kerberos is specified, then the **KRBPlugin** may also be optionally specified. If **KRBPlugin** is not specified, the default plugin IBMkrb5 will be used.

Starting in DB2 Version 9.7 Fix Pack 6 and later, CERTIFICATE authentication is available for connection to DB2 for z/OS Version 10 with APAR PM53450 and later. The CERTIFICATE authentication type is supported starting in DB2 Version 9.7 Fix Pack 6. This authentication type allows you to use SSL client authentication without the need of providing database passwords on the database client. When certificate-based authentication is configured to supply authentication information, a password cannot be specified in any other way (as in the db2dsdriver.cfg configuration file, in the db2cli.ini configuration file, or in the connection string). If CERTIFICATE is specified, then the new label parameter SSLClientLabel must also be specified in the CLI configuration file, db2cli.ini, or in the data server driver configuration file, db2dsdriver.cfg.

---

## AutoCommit CLI/ODBC configuration keyword

Specifies whether the application commits each statement by default.

**db2cli.ini keyword syntax:**

AutoCommit = 1 | 0

**Default setting:**

Each statement is treated as a single, complete transaction.

**Equivalent connection attribute:**

SQL\_ATTR\_AUTOCOMMIT

**Usage notes:**

To be consistent with ODBC, CLI defaults with AutoCommit on, which means each statement is treated as a single, complete transaction. This keyword can provide an alternative default, but will only be used if the application does not specify a value for SQL\_ATTR\_AUTOCOMMIT.

- 1 = SQL\_ATTR\_AUTOCOMMIT\_ON (default)
- 0 = SQL\_ATTR\_AUTOCOMMIT\_OFF

**Note:** Most ODBC applications assume the default of AutoCommit to be on. Extreme care must be used when overriding this default during runtime as the application may depend on this default to operate properly.

This keyword also allows you to specify whether autocommit should be enabled in a Distributed Unit of Work (DUOW) environment. If a connection is part of a coordinated Distributed Unit of Work, and AutoCommit is not set, the default does not apply; implicit commits arising from autocommit processing are suppressed. If AutoCommit is set to 1, and the connection is part of a coordinated Distributed Unit of Work, the implicit commits are processed. This may result in severe performance degradation, and possibly other unexpected results elsewhere in the DUOW system. However, some applications may not work at all unless this is enabled.



A thorough understanding of the transaction processing of an application is necessary, especially applications written by a third party, before applying it to a DUOW environment.

---

### BIDI CLI/ODBC configuration keyword

Specifies the BIDI codepage when we are connected to a DB2 for z/OS.

**db2cli.ini keyword syntax:**

BIDI = *codepage*

**Usage notes:**

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- Database
- Protocol=TCPIP
- Hostname
- Port

---

### BitData CLI/ODBC configuration keyword

Specifies whether binary data types are reported as binary or character data types.

**db2cli.ini keyword syntax:**

BitData = 1 | 0

**Default setting:**

Report FOR BIT DATA and BLOB data types as binary data types.

**Usage notes:**

This option allows you to specify whether ODBC binary data types (SQL\_BINARY, SQL\_VARBINARY, SQL\_LONGVARBINARY, and SQL\_BLOB), are reported as binary type data. IBM DBMSs support columns with binary data types by defining CHAR, VARCHAR, and LONG VARCHAR columns with the FOR BIT DATA attribute. DB2 Database for Linux, UNIX, and Windows will also support binary data via the BLOB data type (in this case it is mapped to a CLOB data type).

Only set BitData = 0 if you are sure that all columns defined as FOR BIT DATA or BLOB contain only character data, and the application is incapable of displaying binary data columns.

- 1 = report FOR BIT DATA and BLOB data types as binary data types (default).
- 0 = report FOR BIT DATA and BLOB data types as character data types.

---

### BlockForNRows CLI/ODBC configuration keyword

Specifies the number of rows of data to be returned in a single fetch.

**db2cli.ini keyword syntax:**

BlockForNRows = <positive integer>

**Default setting:**

The server returns as many rows as can fit in a query block in a single fetch request.

## BlockForNRows CLI/ODBC configuration keyword

### Usage notes:

The BlockForNRows keyword controls the number of rows of data that are returned to the client in a single fetch request. If BlockForNRows is not specified (the default setting), then as many rows of non-LOB data as can fit in a query block are returned from the server. If the result set contains LOB data, then the behavior BlockForNRows yields can be affected by the BlockLobs CLI/ODBC configuration keyword and the server's support for blocking of result sets returning LOB data types.

All LOB data associated with rows that fit completely within a single query block are returned in a single fetch request if:

- BlockForNRows is not specified,
- BlockLobs is set to 1 and
- the server supports blocking of result sets returning LOB data types.

LOB data is described here as being associated with a row, because the LOB data of a result set is itself not contained in the row. Instead, the row contains a reference to the actual LOB data.

If BlockForNRows is set to a positive integer *n*, then *n* rows of data will be returned in a single fetch request. If the result set contains LOB data and the server supports blocking of result sets returning LOB data types, then the LOB data that corresponds to the *n* rows of data will also be returned in the single fetch request. If the result set contains LOB data, but the server does not support blocking of result sets returning LOB data types, then only one row of data, including the LOB data, will be returned in a single fetch request.

---

## BlockLobs CLI/ODBC configuration keyword

Enables LOB blocking fetch against servers that support LOB blocking.

### db2cli.ini keyword syntax:

```
BlockLobs = 0 | 1
```

### Default setting:

Blocking of result sets returning LOB data types is disabled.

### Equivalent statement attribute:

```
SQL_ATTR_BLOCK_LOBS
```

### Usage notes:

Setting BlockLobs to 1 enables all of the LOB data associated with rows that fit completely within a single query block to be returned in a single fetch request, if the server supports LOB blocking. CLI clients which enable BlockLobs = 1 and bind the LOB values directly to buffers can show an increase in memory consumption depending on the amount of data retrieved for one request compared to previous releases. LOB data is described here as being associated with a row, because the LOB data of a result set is itself not contained in the row. Instead, the row contains a reference to the actual LOB data. Therefore, with blocking of result sets returning LOB data types, any rows of the result set that fit completely within the query block (where each row consists of non-LOB data, since LOB data is not stored directly in the row), will have their associated LOB data returned from the server, if the server supports blocking of result sets returning LOB data types.

If the server does not support cursor blocking with LOB columns, then only one row of LOB data will be returned in a single fetch request and the BlockLobs value is ignored. While DB2 does support cursor blocking with LOB columns, other servers may not.

DB2 LUW does not support LOB blocking fetch.

IDS data servers do not support LOB blocking fetch.

---

## CLIPkg CLI/ODBC configuration keyword

Specifies the number of large packages to be generated.

**db2cli.ini keyword syntax:**

CLIPkg = 3 | 4 | ... | 30

**Default setting:**

Three large packages are generated.

**Usage notes:**

This keyword is used to increase the number of sections for SQL statements in CLI/ODBC applications. If it is used, the administrator should explicitly bind the required bind files with the CLIPkg bind option. For client applications, the db2cli.ini file on the client must be updated with this value of CLIPkg. For CLI/JDBC stored procedures, the db2cli.ini file on the server (DB2 Database for Linux, UNIX, and Windows) must be updated with the same value of CLIPkg.

If the value is NOT an integer between 3 and 30, the default will be used without error or warning.

This setting only applies to large packages (containing 384 sections). The number of small packages (containing 64 sections) is 3 and cannot be changed.

It is recommended that you only increase the number of sections enough to run your application as the packages take up space in the database.

---

## CheckForFork CLI/ODBC configuration keyword

Checks for a forked process for each function call.

**db2cli.ini keyword syntax:**

0 | 1

**Default setting:**

CLI does not check for forked processes.

**Usage notes:**

CLI assumes that the process will never be forked. The **CheckForFork** keyword must be set to 1 if applications want to fork while connection and statement handles are allocated in order to avoid interfering with the parent process' active connections.

The SQL\_ATTR\_PROCESSCTL environment attribute can be set to SQL\_PROCESSCTL\_NOTHREAD option by an application to override the **CheckForFork** keyword for that application.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

### ClientAcctStr CLI/ODBC configuration keyword

Allows a CLI application to set the client accounting string that is sent to a database through DB2 Connect or DB2 database products. Applications that do not set the accounting string can use this keyword to provide the client accounting string information.

**db2cli.ini keyword syntax:**

ClientAcctStr = *accounting string*

**Default setting:**

None

**Applicable when:**

Connected to a database using DB2 Connect or DB2 Database for Linux, UNIX, and Windows

**Equivalent environment or connection attribute:**

SQL\_ATTR\_INFO\_ACCTSTR

**Usage notes:**

Note the following conditions:

- Some servers might not be able to handle the entire length of the value and might truncate it.
- DB2 for z/OS and OS/390 servers support a length of up to 200 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the **ClientAcctStr** keyword on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 -9 and the underscore (\_) or period (.).

---

### ClientApplName CLI/ODBC configuration keyword

Allows a CLI application to set the client application name that is sent to a database through DB2 Connect or DB2 database products. Applications that do not specify the application name by default can use this keyword to provide this information.

**db2cli.ini keyword syntax:**

ClientApplName = *application name*

**Default setting:**

None

**Applicable when:**

Connected to a database using DB2 Connect or DB2 for Linux, UNIX, and Windows database products.

**Equivalent environment or connection attribute:**

SQL\_ATTR\_INFO\_APPLNAME

**Usage notes:**

Note the following conditions:

- Some servers might not be able to handle the entire length of the value and might truncate it.

## ClientAppName CLI/ODBC configuration keyword

- DB2 for z/OS and OS/390 servers support a length of up to 32 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the **ClientAppName** keyword on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

---

## ClientBuffersUnboundLOBS CLI/ODBC configuration keyword

Specifies whether LOB data is fetched instead of the LOB locator for LOB columns that have not been bound to application parameters.

### db2cli.ini keyword syntax:

ClientBuffersUnboundLOBS = 0 | 1

### Default setting:

A LOB locator is retrieved instead of the actual LOB data for LOB columns that have not been bound to application parameters.

### Usage notes:

By default, when a result set contains a LOB column that has not been bound to an application parameter, CLI will fetch the corresponding LOB locator rather than the LOB data itself. The application must then use the `SQLGetLength()`, `SQLGetPosition()`, and `SQLGetSubString()` CLI functions to retrieve the LOB data. If the application regularly wants to retrieve the LOB data, then this default two-step process is unnecessary and could decrease performance. In this case, set `ClientBuffersUnboundLOBS = 1` to force DB2 CLI to fetch the LOB data instead of the LOB locator.

Servers that support Dynamic Data Format, also known as progressive streaming, optimize the return of LOB and XML data depending on the actual length of the data. The LOB and XML data can be returned in its entirety, or as an internal token called a progressive reference. CLI manages progressive reference data retrieval.

For applications that are querying data on a server that supports Dynamic Data Format, setting the `LOBCacheSize` keyword sets a threshold that is used to determine if the data is returned in its entirety, or as a progressive reference. If the data has a length greater than the `LOBCacheSize` threshold value, the progressive reference will be returned to CLI to manage, but if the data has a length less than or equal to the `LOBCacheSize` threshold value, the data will be returned in its entirety. Setting `ClientBuffersUnboundLOBS` to 1 is equivalent to setting `LOBCacheSize` to 2147483647 and will force the server to return the data in its entirety rather than as a progressive reference.

---

## ClientEncAlg CLI/ODBC configuration keyword

Specifies the type of encryption algorithm to be used when encrypting user IDs and passwords.

### db2cli.ini keyword syntax:

ClientEncAlg = 1 | 2 | AES

### Default setting:

Any encryption algorithm can be used.

## ClientEncAlg CLI/ODBC configuration keyword

### Applicable when:

Connecting to remote databases.

### Equivalent environment or connection attribute:

SQL\_ATTR\_CLIENT\_ENCALG

### Usage notes:

The values for this keyword are defined as follows:

- 1 - Encrypt the user ID and password using any encryption algorithm.
- 2 - Encrypt the user ID and password using an Advanced Encryption Standard (AES) encryption algorithm.
- AES - Equivalent to 2.

The CLI attribute SQL\_ATTR\_CLIENT\_ENCALG has a similar behavior as the keyword, except when an invalid attribute value is specified an error is returned: CLI0191E Invalid attribute value. The CLI keyword or connection attribute values take precedence over the authentication type specified in the system database directory.

---

## ClientUserID CLI/ODBC configuration keyword

Allows a CLI application to set the client user ID (accounting user ID) that is sent to a database through DB2 Connect or DB2 database products. Applications that do not specify the client user ID by default can use this keyword to provide this information.

### db2cli.ini keyword syntax:

ClientUserID = *userid*

### Default setting:

None

### Applicable when:

Connected to a database using DB2 Connect or DB2 Database for Linux, UNIX, and Windows

### Equivalent environment or connection attribute:

SQL\_ATTR\_INFO\_USERID

### Usage notes:

Do not confuse the client user ID with the authentication user ID. The client user ID is for identification purposes only and is not used for any authorization.

Note the following conditions:

- Some servers might not be able to handle the entire length of the value and might truncate it.
- DB2 for z/OS and OS/390 servers support a length of up to 16 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the **ClientUserID** keyword on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- This user ID is not to be confused with the authentication user ID. This user ID is for identification purposes only and is not used for any authorization.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

---

## ClientWrkStnName CLI/ODBC configuration keyword

Allows a CLI application to set the client workstation name that is sent to a database through DB2 Connect or DB2 database products. Applications that do not specify the client workstation name by default can use this keyword to provide this information.

### db2cli.ini keyword syntax:

ClientWrkStnName = *workstation name*

### Default setting:

In DB2 Version 9.7 Fix pack 6 and later, if **ClientWrkStnName** keyword is not specified, default value that consists of the host name is used. The host name is obtained by gethostname() function call. If host name is not configured or an error is encountered during a gethostname() function call, no value for the **ClientWrkStnName** keyword is sent to the server.

### Applicable when:

Connected to a database using DB2 Connect or DB2 Database for Linux, UNIX, and Windows

### Equivalent environment or connection attribute:

SQL\_ATTR\_INFO\_WRKSTNNAME

### Usage notes:

Note the following conditions:

- Some servers might not be able to handle the entire length of the value and might truncate it.
- DB2 for z/OS and OS/390 servers support a length of up to 18 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the **ClientWrkStnName** keyword on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

---

## ColumnwiseMRI CLI/ODBC configuration keyword

In DB2 Version 9.7 Fix Pack 5 and later fix packs, this keyword specifies whether array input chaining is converted to column-wise array input for DB2 for z/OS servers.

### db2cli.ini keyword syntax:

ColumnwiseMRI = ON | OFF

### Default setting:

Conversion from array input chaining into column-wise array input is disabled.

### Equivalent connection attribute:

SQL\_ATTR\_COLUMNWISE\_MRI

### Usage notes:

The multi-row insert (MRI) feature in DB2 for z/OS expects data to be in column-wise array form. Therefore, you can use this conversion to

## ColumnwiseMRI CLI/ODBC configuration keyword

optimize performance in applications that use the array input chaining feature because the data is sent in a compacted form. The following example shows how to enable this conversion:

```
[dsn-name]  
...  
ColumnwiseMRI=ON
```

The conversion is not performed in certain cases. For details about these cases, see SQL\_ATTR\_COLUMNWISE\_MRI.

This keyword only affects DB2 for z/OS servers.

---

## CommitOnEOF CLI/ODBC configuration keyword

Specifies whether an implicit COMMIT is issued immediately after receiving the last row from a result set. You can free resources as soon as the application receives the entire result set from a cursor by using this keyword.

**db2cli.ini keyword syntax:**

```
CommitOnEOF = 0 | 1
```

**Default setting:**

0

**Equivalent connection attribute:**

SQL\_ATTR\_COMMITONEOF

**Usage notes:**

Starting in DB2 Version 9.7 Fix Pack 5, you can specify whether an implicit COMMIT is issued immediately after receiving the last row from a result set using the **CommitOnEOF** CLI keyword.

You must enable autocommit and the cursor must be read-only and forward-only to be able to take advantage of this optimization.

If stored procedures or applications return multiple result sets, a COMMIT is issued when the last row from the result set of the last cursor is read.

---

## ConcurrentAccessResolution CLI/ODBC configuration keyword

Specifies the concurrent access resolution to use.

**db2cli.ini keyword syntax:**

```
ConcurrentAccessResolution = 0 | 1 | 2 | 3
```

**Default setting:**

DB2 CLI does not supply a prepare option, and the currently committed behavior is determined by the database configuration.

**Applicable when:**

Connected to a database using DB2 Connect or DB2 Database for Linux, UNIX, and Windows

**Equivalent environment or connection attribute:**

SQL\_ATTR\_CONCURRENT\_ACCESS\_RESOLUTION

**Usage notes:**

This keyword specifies a prepare attribute that overrides the default behavior specified for cursor stability (CS) scans.

- 0 = No setting. The client does not supply a prepare option.



## ConcurrentAccessResolution CLI/ODBC configuration keyword

- 1 = Use currently committed semantics. CLI flows "currently committed" on every prepare, which means that the database manager can use the currently committed version of the data for applicable scans when the data is in the process of being updated or deleted. Rows in the process of being inserted can be skipped. This setting applies when the isolation level in effect is Cursor Stability or Read Stability (for Read Stability it skips uncommitted inserts only) and is ignored otherwise. Applicable scans include read-only scans that can be part of a read-only statement as well as a non read-only statement. The settings for the registry variables **DB2\_EVALUNCOMMITTED**, **DB2\_SKIPDELETED**, and **DB2\_SKIPINSERTED** do not apply to scans using currently committed. However, the settings for these registry variables still apply to scans that do not use currently committed.
- 2 = Wait for outcome. CLI flows "wait for outcome" on every prepare, which means that Cursor Stability and higher scans wait for the commit or rollback when encountering data in the process of being updated or deleted. Rows in the process of being inserted are not skipped. The settings for the registry variables **DB2\_EVALUNCOMMITTED**, **DB2\_SKIPDELETED**, and **DB2\_SKIPINSERTED** no longer apply.
- 3 = Skip locked data. CLI flows "skip locked data" on every prepare, which means that currently committed semantics are used and rows in the process of being inserted are skipped. This option is not supported on DB2 Database for Linux, UNIX, and Windows. If specified, this setting is ignored.

For DB2 Database for Linux, UNIX, and Windows, use this keyword to override the default behavior for currently committed that is defined by the **cur\_commit** configuration parameter. For DB2 for z/OS, use this keyword to enable currently committed behavior. There is no equivalent database configuration parameter available on DB2 for z/OS for specifying this behavior.

DB2 z/OS Version 10 only supports INSERT and DELETE operations of currently committed.

---

## ConnectNode CLI/ODBC configuration keyword

Specifies the database partition server to which a connection is to be made.

### db2cli.ini keyword syntax:

```
ConnectNode = integer value from 0 to 999 |  
SQL_CONN_CATALOG_NODE
```

### Default setting:

Database partition server which is defined with port 0 on the machine is used.

### Only applicable when:

Connecting to a partitioned database environment.

### Equivalent connection attribute:

```
SQL_ATTR_CONNECT_NODE
```

### Usage notes:

Used to specify the target database partition server that you want to connect to. Can be set to:

- an integer between 0 and 999
- **SQL\_CONN\_CATALOG\_NODE**

## ConnectNode CLI/ODBC configuration keyword

If this variable is not set, the target defaults to the database partition server that is defined with port 0 on the machine.

**Note:** This keyword does not affect the Control Center. The Control Center always connects to the catalog partition referred to by the `SQL_CONN_CATALOG_NODE` setting.

**Important:** The Control Center and its associated components have been deprecated in Version 9.7 and might be removed in a future release. Start using IBM Data Studio and IBM Optim™ tools. For a mapping between these recommended tools and Control Center tools, see "Table of recommended tools versus Control Center tools" in the *What's New for DB2 Version 9.7* book.

This keyword (or attribute setting) overrides the value of the `DB2NODE` environment variable. Any out of range value specified for this keyword is ignored and the `SQL_CONN_CATALOG_NODE` value is used instead.

---

## ConnectTimeout CLI/ODBC configuration keyword

Specifies the time in seconds to wait for a reply when trying to establish a connection to a server before terminating the attempt and generating a communication timeout.

**db2cli.ini keyword syntax:**

`ConnectTimeout = 0 | 1 | 2 | ... | 32767`

**Default setting:**

The client waits indefinitely for a reply from the server when trying to establish a connection.

**Equivalent connection attribute:**

`SQL_ATTR_LOGIN_TIMEOUT`

**Usage notes:**

If **ConnectTimeout** is set and client reroute is enabled, a connection will be attempted only once to the original server and once to the alternate server. Since the **ConnectTimeout** value is used when attempting to connect to each server, the maximum waiting time will be approximately double the specified value for **ConnectTimeout**. If neither server can be reached within the amount of time specified by the keyword, the following error message will be received:

```
SQL30081N  A communication error has been detected.  Communication
           protocol being used: "TCP/IP".  Communication API being used:
           "SOCKETS".  Location where the error was detected: "<ip address>".
           Communication function detecting the error: "<failing function>".
           Protocol specific error code(s): "<error code>", "*", "*".
           SQLSTATE=08001
```

If **ConnectTimeout** is set and Sysplex exploitation is enabled, a connection will be attempted only once for each of the Sysplex members. Since the **ConnectTimeout** value is used when attempting to connect to each Sysplex member, the maximum waiting time will be approximately equal to the number of Sysplex members, times the amount of time specified by the **ConnectTimeout** keyword.

**ConnectTimeout** only applies to the TCPIP protocol and is not supported for connections to databases cataloged on a SOCKS-enabled TCP/IP node.

## ConnectTimeout CLI/ODBC configuration keyword

A **ConnectTimeout** value explicitly specified in `db2cli.ini` file will take precedence over the `SQL_ATTR_LOGIN_TIMEOUT` during the execution.

---

## ConnectType CLI/ODBC configuration keyword

Controls whether the application is to operate in a remote or distributed unit of work.

**db2cli.ini keyword syntax:**

ConnectType = 1 | 2

**Default setting:**

Remote unit of work.

**Equivalent environment or connection attribute:**

SQL\_ATTR\_CONNECTTYPE

**Usage notes:**

This option allows you to specify the default connect type. The options are:

- 1 = Remote unit of work. Multiple concurrent connections, each with its own commit scope. The concurrent transactions are not coordinated. This is the default.
- 2= Distributed unit of work. Coordinated connections where multiple databases participate under the same distributed unit of work.

The first connection determines the connect type for all other connections that are allocated under the same environment handle.

This keyword takes precedence over the environment or connection attribute.

---

## CurrentFunctionPath CLI/ODBC configuration keyword

Specifies the schema used to resolve function references and data type references in dynamic SQL statements.

**db2cli.ini keyword syntax:**

CurrentFunctionPath = *current\_function\_path*

**Default setting:**

See description below.

**Usage notes:**

This keyword defines the path used to resolve function references and data type references that are used in dynamic SQL statements. It contains a list of one or more schema-names, where schema-names are enclosed in double quotes and separated by commas.

The default value is "SYSIBM","SYSFUN",X where X is the value of the USER special register delimited by double quotes. The schema SYSIBM does not need to be specified. If it is not included in the function path, then it is implicitly assumed as the first schema.

This keyword is used as part of the process for resolving unqualified function and stored procedure references that may have been defined in a schema name other than the current user's schema. The order of the schema names determines the order in which the function and procedure names will be resolved.

### CurrentImplicitXMLParseOption CLI/ODBC configuration keyword

Sets the value of the CURRENT IMPLICIT XMLPARSE OPTION special register.

**db2cli.ini keyword syntax:**

```
CurrentImplicitXMLParseOption = 'STRIP WHITESPACE' | 'PRESERVE WHITESPACE'
```

**Default setting:**

Whitespace is stripped during implicit non-validating parsing.

**Equivalent connection attribute:**

```
SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION
```

**Usage notes:**

Setting this keyword issues the SET CURRENT IMPLICIT XMLPARSE OPTION statement after every connection to a database. By default, this statement is not issued.

The SET CURRENT IMPLICIT XMLPARSE OPTION statement sets the CURRENT IMPLICIT XMLPARSE OPTION special register, which controls whether white space is stripped or preserved during implicit non-validating parsing.

CurrentImplicitXMLParseOption does not affect explicit parsing with the XMLPARSE function.

The supported settings for CurrentImplicitXMLParseOption are:

- STRIP WHITESPACE - white space is removed when an XML document is implicitly parsed. This is the default setting.
- PRESERVE WHITESPACE - white space is preserved when an XML document is implicitly parsed.

---

### CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword

Sets the value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.

**db2cli.ini keyword syntax:**

```
CurrentMaintainedTableTypesForOpt = ALL | FEDERATED_TOOL | NONE | SYSTEM | USER | <list>
```

**Default setting:**

System-maintained refresh-deferred materialized query tables are considered in the optimization of a query.

**Usage notes:**

This keyword defines the default value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. The value of the special register affects the types of tables which are considered in the optimization of a query. Refer to the SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION SQL statement for details on the supported settings of ALL, FEDERATED\_TOOL, NONE, SYSTEM, or USER. The <list> option represents a combination of the supported settings, however, ALL and NONE cannot be specified with any other value, and the same value cannot be specified more than once. Separate each value in the list with a comma, for example:

## CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword

CurrentMaintainedTableTypesForOpt = SYSTEM,USER

---

## CURRENTOPTIMIZATIONPROFILE CLI/ODBC configuration keyword

Specifies the optimization profile used in a SET CURRENT OPTIMIZATION PROFILE statement upon a successful connection.

**db2cli.ini keyword syntax:**

**CURRENTOPTIMIZATIONPROFILE** =NULL|*optimization-profile-name*

**Default setting:**

NULL

**Usage notes:**

**NULL**

Sets the register to the null value.

*optimization-profile-name*

Sets the CURRENT OPTIMIZATION PROFILE special register to the name of an optimization profile. If optimization-profile-name is unqualified, then the default schema qualification is applied.

### Examples

If a DB2CLI.INI file has the following entry, after each successful connection to the "Rochester" database, the CLI client would issue the command SET CURRENT OPTIMIZATION PROFILE = "Hamid"."RochesterProfile".

```
[Rochester]
CURRENTOPTIMIZATIONPROFILE="Hamid"."RochesterProfile"
```

In this example, the optimization profile name is delimited by quotation marks because it contains lower case characters.

---

## CurrentPackagePath CLI/ODBC configuration keyword

Issues 'SET CURRENT PACKAGE PATH = *schema1, schema2, ...*' after every connection.

**db2cli.ini keyword syntax:**

**CurrentPackagePath** = *schema1, schema2, ...*

**Default setting:**

The clause is not appended.

**Equivalent connection attribute:**

SQL\_ATTR\_CURRENT\_PACKAGE\_PATH

**Usage notes:**

When set, this option issues the command "SET CURRENT PACKAGE PATH = *schema1, schema2, ...*" after every connection to the database. This setting specifies the list of schema names (collection identifiers) that will be searched when there is a package from a different schema.

This keyword is best suited for use with ODBC static processing applications, rather than CLI applications.

### CurrentPackageSet CLI/ODBC configuration keyword

Issues the SET CURRENT PACKAGESET statement after every connection.

**db2cli.ini keyword syntax:**

CurrentPackageSet = *schema name*

**Default setting:**

The clause is not appended.

**Equivalent connection attribute:**

SQL\_ATTR\_CURRENT\_PACKAGE\_SET

**Usage notes:**

This option issues the SET CURRENT PACKAGESET SQL statement with the CurrentPackageSet value after every connection to a database. By default this clause is not appended.

The SET CURRENT PACKAGESET SQL statement sets the schema name (collection identifier) that is used to select the package to use for subsequent SQL statements.

CLI/ODBC applications issue dynamic SQL statements. Using this option you can control the privileges used to run these statements:

- Choose a schema to use when running SQL statements from CLI/ODBC applications.
- Ensure the objects in the schema have the desired privileges and then rebind accordingly.
- Set the CurrentPackageSet option to this schema.

The SQL statements from the CLI/ODBC applications will now run under the specified schema and use the privileges defined there.

The following package set names are reserved: NULLID, NULLIDR1, NULLIDRA.

If both the Reopt and CurrentPackageSet keywords are specified, CurrentPackageSet takes precedence.

---

### CurrentRefreshAge CLI/ODBC configuration keyword

Sets the value of the CURRENT REFRESH AGE special register.

**db2cli.ini keyword syntax:**

CurrentRefreshAge = 0 | ANY | positive integer

**Default setting:**

Only materialized query tables defined with REFRESH IMMEDIATE may be used to optimize the processing of a query.

**Usage notes:**

Setting this keyword sets the value of the CURRENT REFRESH AGE special register.

---

## CurrentSQLID CLI/ODBC configuration keyword

Specifies the ID used in a SET CURRENT SQLID statement sent to the DBMS upon a successful connection.

**db2cli.ini keyword syntax:**

CurrentSQLID = *current\_sqlid*

**Default setting:**

No statement is issued.

**Only applicable when:**

connecting to those DB2 DBMS's where SET CURRENT SQLID is supported.

**Usage notes:**

Upon a successful connection, if this option is set, a SET CURRENT SQLID statement is sent to the DBMS. This allows the end user and the application to name SQL objects without having to qualify them by schema name.

---

## CurrentSchema CLI/ODBC configuration keyword

Specifies the schema used in a SET CURRENT SCHEMA statement upon a successful connection.

**db2cli.ini keyword syntax:**

CurrentSchema = *schema name*

**Default setting:**

No statement is issued.

**Usage notes:**

Upon a successful connect, if this option is set, a SET CURRENT SCHEMA statement is sent to the DBMS. This allows the end user or application to name SQL objects without having to qualify them by schema name.

---

## CursorHold CLI/ODBC configuration keyword

Controls the effect of a transaction completion on open cursors.

**db2cli.ini keyword syntax:**

CursorHold = 1 | 0

**Default setting:**

Selected--Cursors are not destroyed.

**Equivalent statement attribute:**

SQL\_ATTR\_CURSOR\_HOLD

**Usage notes:**

This option controls the effect of a transaction completion on open cursors.

- 1 = SQL\_CURSOR\_HOLD\_ON, the cursors are not destroyed when the transaction is committed (default).
- 0 = SQL\_CURSOR\_HOLD\_OFF, the cursors are destroyed when the transaction is committed.

## CursorHold CLI/ODBC configuration keyword

**Note:** Cursors are always closed when transactions are rolled back.

This option affects the result returned by `SQLGetInfo()` when called with `SQL_CURSOR_COMMIT_BEHAVIOR` or `SQL_CURSOR_ROLLBACK_BEHAVIOR`. The value of `CursorHold` is ignored if connecting to DB2 Server for VSE & VM where cursor with hold is not supported.

You can use this option to tune performance. It can be set to `SQL_CURSOR_HOLD_OFF` (0) if you are sure that your application:

1. Does not have behavior that is dependent on the `SQL_CURSOR_COMMIT_BEHAVIOR` or the `SQL_CURSOR_ROLLBACK_BEHAVIOR` information returned via `SQLGetInfo()`, and
2. Does not require cursors to be preserved from one transaction to the next.

The DBMS will operate more efficiently with `CursorHold` disabled, as resources no longer need to be maintained after the end of a transaction.

---

## CursorTypes CLI/ODBC configuration keyword

Specifies which cursor types are permitted.

**db2cli.ini keyword syntax:**

`CursorTypes = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7`

**Default setting:**

Forward-only, static, keyset-driven, and dynamic cursors are supported if the server supports them.

**Usage notes:**

The `CursorTypes` keyword is a bitmask that indicates what types of cursors an application can open:

- 0x0 - forward-only (can always be opened)
- 0x1 - static
- 0x2 - keyset-driven
- 0x4 - dynamic

For example,

- to prevent applications from opening dynamic scrollable cursors, set `CursorTypes` to 3.
- to allow applications to open only non-scrollable cursors, set `CursorTypes` to 0.

This keyword only affects calls made to the following CLI functions:

- `SQLBulkOperations()`
- `SQLExecDirect()`
- `SQLExecute()`
- `SQLFetchScroll()`
- `SQLPrepare()`
- `SQLSetPos()`

---

## DB2Degree CLI/ODBC configuration keyword

Sets the degree of parallelism for the execution of SQL statements.



### db2cli.ini keyword syntax:

DB2Degree = 0 | integer value from 1 to 32767 | ANY

### Default setting:

No SET CURRENT DEGREE statement is issued.

### Only applicable when:

connecting to a cluster database system.

### Usage notes:

If the value specified is anything other than 0 (the default) then CLI will issue the following SQL statement after a successful connection:

```
SET CURRENT DEGREE value
```

This specifies the degree of parallelism for the execution of the SQL statements. The database manager will determine the degree of parallelism if you specify ANY.

---

## DB2Explain CLI/ODBC configuration keyword

Determines whether Explain snapshot and/or Explain table information will be generated by the server.

### db2cli.ini keyword syntax:

DB2Explain = 0 | 1 | 2 | 3

### Default setting:

Neither Explain snapshot nor Explain table information will be generated by the server.

### Equivalent connection attribute:

SQL\_ATTR\_DB2EXPLAIN

### Usage notes:

This keyword determines whether Explain snapshot and/or Explain table information will be generated by the server.

- 0 = both off (default)

A 'SET CURRENT EXPLAIN SNAPSHOT=NO' and a 'SET CURRENT EXPLAIN MODE=NO' statement will be sent to the server to disable both the Explain snapshot and the Explain table information capture facilities.

- 1 = Only Explain snapshot facility on

A 'SET CURRENT EXPLAIN SNAPSHOT=YES' and a 'SET CURRENT EXPLAIN MODE=NO' statement will be sent to the server to enable the Explain snapshot facility, and disable the Explain table information capture facility.

- 2 = Only Explain table information capture facility on

A 'SET CURRENT EXPLAIN MODE=YES' and a 'SET CURRENT EXPLAIN SNAPSHOT=NO' will be sent to the server to enable the Explain table information capture facility and disable the Explain snapshot facility.

- 3 = Both on

A 'SET CURRENT EXPLAIN MODE=YES' and a 'SET CURRENT EXPLAIN SNAPSHOT=YES' will be sent to the server to enable both the Explain snapshot and the Explain table information capture facilities.

## DB2Explain CLI/ODBC configuration keyword

Explain information is inserted into Explain tables, which must be created before the Explain information can be generated. The current authorization ID must have INSERT privilege for the Explain tables.

### Remarks

Starting in Version 9.7 Fix Pack 3 and later fix packs, DB2 z/OS server supports only 0 (OFF) and 2 (Table) values in DB2Explain settings as only explain mode information is available. If the DB2Explain keyword is attempted to set against the data server which do not support it, the application receives an error "CLI0150E Driver not capable".

---

## DB2NETNamedParam CLI/ODBC configuration keyword

Specifies if named parameters are used by DB2 .NET applications.

### db2cli.ini keyword syntax:

DB2NETNamedParam = 0 | 1

### Default setting:

The DB2 .NET Data Provider recognizes named parameters as parameters, but ignores positioned parameters, in SQL statements.

### Usage notes:

By default, the DB2 .NET Data Provider processes tokens in an SQL statement with the format "@<paramname>" as named parameters and ignores any positioned parameters, where positioned parameters are specified with a '?' character or a colon followed by a name (:name).

The following is an example of a query that contains a named parameter:

```
SELECT * FROM T1 WHERE C1 = @param1
```

This is an example of a query that contains a positioned parameter:

```
SELECT * FROM T1 WHERE C1 = ?
```

Specify 0 to indicate that only positioned parameters will be recognized as parameters in SQL statements. This setting can improve application performance by reducing the overhead required to process named parameters.

---

## DB2Optimization CLI/ODBC configuration keyword

Sets the query optimization level.

### db2cli.ini keyword syntax:

DB2Optimization = *integer value from 0 to 9*

### Default setting:

No SET CURRENT QUERY OPTIMIZATION statement issued.

### Usage notes:

If this option is set then CLI will issue the following SQL statement after a successful connection:

```
SET CURRENT QUERY OPTIMIZATION positive number
```

This specifies the query optimization level at which the optimizer should operate the SQL queries.

---

### DBAlias CLI/ODBC configuration keyword

Specifies the database alias for a data source name (DSN) that is greater than 8 characters.

**db2cli.ini keyword syntax:**

DBAlias = *dbalias*

**Default setting:**

Use the DB2 database alias as the ODBC Data Source Name.

**Usage notes:**

The DSN is the name, enclosed in square brackets, that denotes the section header in the `db2cli.ini` file. Typically, this section header is the database alias name that has a maximum length of 8 bytes. If you want to use a longer, more meaningful name, you can place the longer name in the section header, and set this keyword value to the database alias that is used on the **CATALOG** command. Here is an example:

```
; The much longer name maps to an 8 single byte character dbalias  
[MyMeaningfulName]  
DBAlias=DB2DBT10
```

You can specify `[MyMeaningfulName]` as the name of the data source on connect while the actual database alias is `DB2DBT10`.

In DB2 Version 9.7 Fix Pack 5 and later fix packs, if you specify a value in the **DBAlias** keyword with the database keyword in the `db2cli.ini` file, applications that try to connect to the database or DSN that matches this value do not receive an error.

---

### DBName CLI/ODBC configuration keyword

Specifies the database name to reduce the time it takes for the application to query z/OS or OS/390 table information.

**db2cli.ini keyword syntax:**

DBName = *dbname*

**Default setting:**

Do not filter on the DBNAME column.

**Only applicable when:**

connecting to DB2 for z/OS and OS/390.

**Usage notes:**

This option is only used when connecting to DB2 for z/OS and OS/390, and only if (*base*) table catalog information is requested by the application. If a large number of tables exist in the z/OS or OS/390 subsystem, a *dbname* can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application.

## DBName CLI/ODBC configuration keyword

If this option is set then the statement `IN DATABASE dbname` will be appended to various statements such as `CREATE TABLE`.

This value maps to the `DBNAME` column in the z/OS or OS/390 system catalog tables. If no value is specified, or if views, synonyms, system tables, or aliases are also specified via `TableType`, only table information will be restricted; views, aliases, and synonyms are not restricted with `DBName`. It can be used in conjunction with `SchemaList`, and `TableType` to further limit the number of tables for which information will be returned.

---

## DSN CLI/ODBC configuration keyword

Sets the name of a data source as returned by `SQLDataSources` or the data sources dialog box of `SQLDriverConnect`.

### db2cli.ini keyword syntax:

You can not set this keyword in the `db2cli.ini` file.

You can specify the value of this keyword in the connection string in `SQLDriverConnect` like this:

`DSN = database name`

---

## Database CLI/ODBC configuration keyword

Specifies the database on the server to connect to when using a File DSN.

### db2cli.ini keyword syntax:

`Database = database name`

### Default setting:

None

### Only applicable when:

Protocol set to TCPIP

### Usage notes:

When using a File DSN you must use this option to specify the database on the server to connect to. This value has nothing to do with any database alias name specified on the client, it must be set to the database name on the server itself.

This setting is only considered when the Protocol option is set to TCPIP.

---

## DateTimeStringFormat CLI/ODBC configuration keyword

Specifies the format to use when inserting date or time data into character columns.

### db2cli.ini keyword syntax:

`DateTimeStringFormat = JIS | ISO | EUR | USA`

### Default setting:

The JIS format is used when date or time data is inserted into character columns.

### Usage notes:

## DateTimeStringFormat CLI/ODBC configuration keyword

The DateTimeStringFormat keyword controls the format in which date or time data is inserted into character columns. This setting affects the insertion of SQL\_C\_TYPE\_DATE, SQL\_C\_TYPE\_TIME, or SQL\_C\_TYPE\_TIMESTAMP, or SQL\_C\_TIMESTAMP\_EXT data into the following column types:

- SQL\_CHAR
- SQL\_VARCHAR
- SQL\_LONGVARCHAR
- SQL\_CLOB

This keyword also affects the format of date or time columns that are retrieved into character strings. For example, retrieving data from an SQL\_TYPE\_TIMESTAMP column into an SQL\_C\_CHAR string will be affected by the setting of this keyword.

The four setting values are as follows:

Format	Date	Time	Timestamp
JIS	yyyy-mm-dd	hh:mm:ss	yyyy-mm-dd hh:mm:ss.ffffffffffff
ISO	yyyy-mm-dd	hh.mm.ss	yyyy-mm-dd- hh.mm.ss.ffffffffffff
EUR	dd.mm.yyyy	hh.mm.ss	yyyy-mm-dd hh:mm:ss.ffffffffffff*
USA	mm/dd/yyyy	hh:mm AM or PM	yyyy-mm-dd hh:mm:ss.ffffffffffff*

\*Timestamps will take the default format if EUR or USA is specified. The default format is JIS.

---

## DecimalFloatRoundingMode CLI/ODBC configuration keyword

Sets the rounding mode when working with servers that support the DECFLOAT SQL type.

**db2cli.ini keyword syntax:**

DecimalFloatRoundingMode = 0 | 1 | 2 | 3 | 4

**Default setting:**

0 (Half even rounding mode)

**Equivalent connection attribute:**

SQL\_ATTR\_DECFLOAT\_ROUNDING\_MODE

**Usage notes:**

The decimal float rounding mode determines what type of rounding will be used if a value is put into a DECFLOAT variable or column but the value has more digits than are allowed in the DECFLOAT data type. This can occur when inserting, updating, selecting, converting from another type, or as the result of a mathematical operation.

The value of SQL\_ATTR\_DECFLOAT\_ROUNDING\_MODE determines the decimal float rounding mode that will be used for new connections unless another mode is specified by a connection attribute for that connection. For any given connection both CLI and DB2 will use the same decimal float rounding mode for all action initiated as part of that connection.

## DecimalFloatRoundingMode CLI/ODBC configuration keyword

When your applications are connecting to a DB2 Database for Linux, UNIX, and Windows Version 9.5 server, you must set the decimal float rounding mode on the database client to the same mode that is set on the server. If you set the decimal float rounding mode on the client to a value that is different from the decimal float rounding mode that is set on the database server, the database server will return SQL0713N on connection.

The settings correspond to these decimal float rounding modes:

- 0 = Half even (default)
- 1 = Half up
- 2 = Down
- 3 = Ceiling
- 4 = Floor

The different modes are:

### Half even (default)

In this mode CLI and DB2 use the number that will fit in the target variable and that is closest to the original value. If two numbers are equally close, they use the one that is even. This mode produces the smallest rounding errors over large amounts of data.

### Half up

In this mode CLI and DB2 use the number that will fit in the target variable and that is closest to the original value. If two numbers are equally close, they use the one that is greater than the original value.

**Down** In this mode CLI and DB2 use the number that will fit in the target variable and that is closest to the original value and for which the absolute value is not greater than the absolute value of the original value. You can also think of this as rounding toward zero or as using ceiling for negative values and using floor for positive values.

### Ceiling

In this mode CLI and DB2 use the smallest number that will fit in the target variable and that is greater than or equal to the original value.

**Floor** In this mode CLI and DB2 use the largest number that will fit in the target variable and that is less than or equal to the original value.

This attribute is not supported when accessing IDS data servers.

---

## DeferredPrepare CLI/ODBC configuration keyword

Minimizes network flow by combining the PREPARE request with the corresponding execute request.

### db2cli.ini keyword syntax:

DeferredPrepare = 0 | 1

### Default setting:

The prepare request will be delayed until the execute request is sent.

### Equivalent statement attribute:

SQL\_ATTR\_DEFERRED\_PREPARE

### Usage notes:

Defers sending the PREPARE request until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance.

- 0 = SQL\_DEFERRED\_PREPARE\_OFF. The PREPARE request will be executed the moment it is issued.
- 1 = SQL\_DEFERRED\_PREPARE\_ON (default). Defer the execution of the PREPARE request until the corresponding execute request is issued.

If the target DBMS does not support deferred prepare, the client disables deferred prepare for that connection.

**Note:** When deferred prepare is enabled, the row and cost estimates normally returned in the SQLERRD(3) and SQLERRD(4) of the SQLCA of a PREPARE statement may become zeros. This may be of concern to users who want to use these values to decide whether or not to continue the SQL statement.

---

## DescribeCall CLI/ODBC configuration keyword

Determines when stored procedure arguments are described.

### db2cli.ini keyword syntax:

DescribeCall = 1 | -1

### Default setting:

DB2 CLI does not request stored procedure argument describe information when it prepares a CALL statement.

### Equivalent connection attribute:

SQL\_ATTR\_DESCRIBE\_CALL

### Usage notes:

By default, CLI does not request input parameter describe information when it prepares a CALL statement. If an application has correctly bound parameters to a statement, then this describe information is unnecessary and not requesting it improves performance.

The option values are:

- 1 = SQL\_DESCRIBE\_CALL\_BEFORE. CLI always requests describe information from the server, ignoring the binding information provided by the application. Setting DescribeCall to 1 will also set DeferredPrepare to 0 which means that describe information will also be requested for dynamic SQL statements. Note that setting DeferredPrepare to 0 will not set DescribeCall to 1.
- -1 = SQL\_DESCRIBE\_CALL\_DEFAULT (default). CLI does not request describe information from the server and uses the binding information provided by the application. If the CALL statement execution fails, then the CLI error recovery logic requests input parameter describe information from the server and issues the CALL statement again.

---

## DescribeInputOnPrepare CLI/ODBC configuration keyword

Enables or disables the request for describe information when an SQL statement is prepared.

### db2cli.ini keyword syntax:

DescribeInputOnPrepare = 0 | 1

## DescribeInputOnPrepare CLI/ODBC configuration keyword

### Default setting:

Do not request describe information when preparing an SQL statement.

### Usage notes:

By default, CLI does not request input parameter describe information when it prepares an SQL statement. If an application has correctly bound parameters to a statement, then this describe information is unnecessary and not requesting it improves performance. If, however, parameters have not been correctly bound, then statement execution will fail and cause the CLI error recovery retry logic to request input parameter describe information. The result is an additional server request and reduced performance, compared to if the describe information had been requested with the prepare. Setting `DescribeInputOnPrepare` to 1 causes the input describe information to be requested with the prepare. This setting may improve performance for applications which rely heavily on the CLI retry logic to recover from application binding errors.

---

## DescribeOutputLevel CLI/ODBC configuration keyword

Sets the level of output column describe information that is requested by the CLI driver during prepare or describe requests.

### db2cli.ini keyword syntax:

`DescribeOutputLevel = 0 | 1 | 2 | 3`

### Default setting:

Request the describe information listed in level 2 of Table 159 on page 357.

### Equivalent connection attribute:

`SQL_ATTR_DESCRIBE_OUTPUT_LEVEL`

### Usage notes:

This keyword controls the amount of information the CLI driver requests on a prepare or describe request. By default, when the server receives a describe request, it returns the information contained in level 2 of Table 159 on page 357 for the result set columns. An application, however, might not need all of this information or might need additional information. Setting the `DescribeOutputLevel` keyword to a level that suits the needs of the client application might improve performance because the describe data transferred between the client and server is limited to the minimum amount that the application requires. If the `DescribeOutputLevel` setting is set too low, it might impact the functionality of the application (depending on the application's requirements). The CLI functions to retrieve the describe information might not fail in this case, but the information returned might be incomplete. Supported settings for `DescribeOutputLevel` are:

- 0 - no describe information is returned to the client application
- 1 - describe information categorized in level 1 (see Table 159 on page 357) is returned to the client application
- 2 - (default) describe information categorized in level 2 (see Table 159 on page 357) is returned to the client application
- 3 - describe information categorized in level 3 (see Table 159 on page 357) is returned to the client application

The following table lists the fields that form the describe information that the server returns when it receives a prepare or describe request. These



## DescribeOutputLevel CLI/ODBC configuration keyword

fields are grouped into levels, and the **DescribeOutputLevel** CLI/ODBC configuration keyword controls which levels of describe information the CLI driver requests.

### Note:

1. Not all levels of describe information are supported by all DB2 servers. All levels of describe information are supported on the following DB2 servers: DB2 for Linux, UNIX, and Windows Version 8 and later, DB2 for z/OS Version 8 and later, and DB2 for i5/OS® Version 5 Release 3 and later. All other DB2 servers support only the 2 or 0 setting for **DescribeOutputLevel**.
2. The default behavior allows CLI to promote the level to 3 if the application asks for describe information that was not initially retrieved using the default level 2. This might result in two network flows to the server. If an application uses this keyword to explicitly set a describe level, then no promotion will occur. Therefore, if the keyword is used to set the describe level to 2, CLI will not promote to level 3 even if the application asks for extended information.

Table 159. Levels of describe information

Level 1	Level 2	Level 3
SQL_DESC_COUNT SQL_COLUMN_COUNT SQL_DESC_TYPE SQL_DESC_CONCISE_TYPE SQL_COLUMN_LENGTH SQL_DESC_OCTET_LENGTH SQL_DESC_LENGTH SQL_DESC_PRECISION SQL_COLUMN_PRECISION SQL_DESC_SCALE SQL_COLUMN_SCALE SQL_DESC_DISPLAY_SIZE SQL_DESC_NULLABLE SQL_COLUMN_NULLABLE SQL_DESC_UNSIGNED SQL_DESC_SEARCHABLE SQL_DESC_LITERAL_SUFFIX SQL_DESC_LITERAL_PREFIX SQL_DESC_CASE_SENSITIVE SQL_DESC_FIXED_PREC_SCALE	all fields of level 1 and: SQL_DESC_NAME SQL_DESC_LABEL SQL_COLUMN_NAME SQL_DESC_UNNAMED SQL_DESC_TYPE_NAME SQL_DESC_DISTINCT_TYPE SQL_DESC_REFERENCE_TYPE SQL_DESC_STRUCTURED_TYPE SQL_DESC_USER_TYPE SQL_DESC_LOCAL_TYPE_NAME SQL_DESC_USER_DEFINED_ TYPE_CODE	all fields of levels 1 and 2 and: SQL_DESC_BASE_COLUMN_NAME SQL_DESC_UPDATABLE SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_SCHEMA_NAME SQL_DESC_CATALOG_NAME SQL_DESC_TABLE_NAME SQL_DESC_BASE_TABLE_NAME

---

## DescribeParam CLI/ODBC configuration keyword

Enables or disables the SQLDescribeParam() function.

### db2cli.ini keyword syntax:

DescribeParam = 0 | 1

### Default setting:

The SQLDescribeParam() function is enabled.

### Usage notes:

When set to 1 (default), SQLDescribeParam() is enabled and SQLGetFunctions() will return SQLDescribeParam() as supported.

## DescribeParam CLI/ODBC configuration keyword

When set to 0, SQLDescribeParam() is disabled. If SQLDescribeParam() is called, CLI0150E will be returned. SQLGetFunctions() will return SQLDescribeParam() as not supported.

---

## DiagLevel CLI/ODBC configuration keyword

Sets the diagnostic level.

**db2cli.ini keyword syntax:**

DiagLevel = 0 | 1 | 2 | 3 | 4

**Default setting:**

3

**Usage notes:**

This can be set in the [COMMON] section of the db2cli.ini file only.

This is applicable only at Environment Handle allocation time for an entire process.

This is equivalent to the database manager parameter DIAGLEVEL.

---

## DiagPath CLI/ODBC configuration keyword

Sets the path of the **db2diag** log files.

**db2cli.ini keyword syntax:**

DiagPath = *existing directory*

**Default setting:**

The default value is the db2dump directory on UNIX and Linux operating systems, and the db2 directory on Windows operating systems.

**Usage notes:**

This can be set in the [COMMON] section of the db2cli.ini file only.

This is equivalent to the database manager parameter DIAGPATH.

---

## DisableKeysetCursor CLI/ODBC configuration keyword

Disables keyset-driven scrollable cursors.

**db2cli.ini keyword syntax:**

DisableKeysetCursor = 0 | 1

**Default setting:**

Keyset-driven scrollable cursors are returned when requested.

**Usage notes:**

When set to 1, this keyword forces the CLI driver to return a static cursor to the application, even if the application has requested a keyset-driven scrollable cursor. The default setting (0) causes keyset-driven cursors to be returned when the application requests them. This keyword can be used to restore behavior before scrollable cursors were supported.

---

## DisableMultiThread CLI/ODBC configuration keyword

Disables multithreading.

## DisableMultiThread CLI/ODBC configuration keyword

### db2cli.ini keyword syntax:

DisableMultiThread = 0 | 1

### Default setting:

Multithreading is enabled.

### Usage notes:

The CLI/ODBC driver is capable of supporting multiple concurrent threads.

This option is used to enable or disable multi-thread support.

- 0 = Multithreading is enabled (default).
- 1 = Disable multithreading.

If multithreading is disabled then all calls for all threads will be serialized at the process level. Use this setting for multithreaded applications that require serialized behavior.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## DisableUnicode CLI/ODBC configuration keyword

Disables underlying Unicode support.

### db2cli.ini keyword syntax:

DisableUnicode = <not set> | 0 | 1

### Default setting:

Unicode support is enabled.

### Usage notes:

With Unicode support enabled, and when called by a Unicode application, CLI will attempt to connect to the database using the best client code page possible to ensure there is no unnecessary data loss due to code page conversion. This may increase the connection time as code pages are exchanged, or may cause code page conversions on the client that did not occur before this support was added.

If an application is Unicode (the SQL\_ATTR\_ANSI\_APP connection attribute is set to SQL\_AA\_FALSE, or the connection occurred with SQLConnectW()), then the **DisableUnicode** keyword can be used to effect three different connection behaviors:

- **DisableUnicode** is not set in the db2cli.ini file: If the target database supports Unicode, CLI will connect in Unicode code pages (1208 and 1200). Otherwise, CLI will connect in the application code page.
- **DisableUnicode=0** is set: CLI always connects in Unicode, whether or not the target database supports Unicode.
- **DisableUnicode=1** is set: CLI always connects in the application code page, whether or not the target database supports Unicode.

---

## EnableNamedParameterSupport CLI/ODBC configuration keyword

Specifies whether named parameter processing is enabled.

### db2cli.ini keyword syntax:

EnableNamedParameterSupport = TRUE | FALSE

## EnableNamedParameterSupport CLI/ODBC configuration keyword

**Default setting:**

Named parameter support is off (FALSE).

**Equivalent environment or connection attribute:**

None

**Usage notes:**

Named parameter processing allows applications to use named parameters (for example, :name) in addition to traditional unnamed parameter markers that are represented by a question mark (?). The following options are available to enable or disable named parameter support:

- TRUE - Named parameter processing is enabled. For DB2for Linux, UNIX, and WindowsVersion 9.7 and later, CLI sends the statement text as it is to the server for processing. For all other servers, CLI substitutes the statement text by replacing named parameters with question marks (?) before sending the statement to the server for processing.
- FALSE - Named parameter processing is off, and CLI does not process the parameter markers.

There is no support for the ability to bind by name. CLI processes anything that matches a valid parameter marker, and treats it as if it is a normal parameter marker represented by a question mark (?).

---

## FET\_BUF\_SIZE CLI/ODBC configuration keyword

Specifies the default query block size to optimize the data flow.

**db2cli.ini keyword syntax:**

FET\_BUF\_SIZE = 64K | 96K | 128K | 160K | 192K | 224K | 256K

**Default setting:**

FET\_BUF\_SIZE = 64K

**Equivalent connection attribute:**

SQL\_ATTR\_FET\_BUF\_SIZE

**Usage notes:**

CLI allows query block size only in multiples of 32K (that is 64K, 96K, 128K, 160K, 192K, 224K, and 256K). CLI applications round up any other values in the range of 64K-256K, to the next nearest 32K boundary.

Application can obtain the value it has set for this attribute using SQLGetConnectAttr(). If application has not set any value, the default query block size is returned.

---

## FileDSN CLI/ODBC configuration keyword

Specifies a DSN file from which a connection string will be built for the data source.

**db2cli.ini keyword syntax:**

You can not set this keyword in the db2cli.ini file.

You can specify the value of this keyword in the connection string in SQLDriverConnect like this:

FileDSN = *file name*

---

## FloatPrecRadix CLI/ODBC configuration keyword

Forces the NUM\_PREC\_RADIX value of a floating point type to be 2 or 10.

## FloatPrecRadix CLI/ODBC configuration keyword

### db2cli.ini keyword syntax:

FloatPrecRadix = 2 | 10

### Default setting:

Report the NUM\_PREC\_RADIX as 2 for floating point types, as they have a base of 2, not 10.

### Usage notes:

The NUM\_PREC\_RADIX value represents a data type's base. Binary numbers, such as floating point numbers, have a base of 2, and integers have a base of 10. An application may expect all values in the COLUMN\_SIZE field to represent the maximum number of digits, which assumes a NUM\_PREC\_RADIX value of 10. However, for floating point numeric types, the NUM\_PREC\_RADIX is 2, in which case the COLUMN\_SIZE will report the number of bits in the data type's representation, rather than the maximum number of digits.

FloatPrecRadix can force the NUM\_PREC\_RADIX to be reported as 10 for floating point data types, in which case the COLUMN\_SIZE will report the maximum number of digits.

The FloatPrecRadix keyword affects SQLColumns(), SQLGetDescField() (for the SQL\_DESC\_NUM\_PREC\_RADIX field), SQLGetTypeInfo(), SQLProcedureColumns(), and SQLSpecialColumns().

---

## GetDataLobNoTotal CLI/ODBC configuration keyword

Causes SQLGetData() to fetch column data in pieces of specified size (in bytes) instead of fetching column data all at once.

### db2cli.ini keyword syntax:

**GetDataLobNoTotal** = *positive integer*

### Usage notes:

SQLGetData() retrieves data for a single column in the current row of the result set. The first call to SQLGetData() does the following:

- Fetches all the data from the database server to the client, which requires allocating memory on the client for the data
- Applies a codepage conversion to that data, if required
- Calculates the total length of the converted data
- Returns the length of the converted data to the client application

When the data is very large, allocating memory for the data on the client might fail. You can avoid this potential memory allocation problem, by using the **GetDataLobNoTotal** keyword.

When you set the **GetDataLobNoTotal** keyword, SQLGetData() does not fetch all the data for the given column on the first call. Instead, SQLGetData() fetches enough data to fill the buffer on the client, as specified by the value of **GetDataLobNoTotal**, and returns SQL\_NO\_TOTAL (-4) if there is more data to be fetched from the server. You can call SQLGetData() as many times as needed to fetch all the data. When all the data has been fetched, SQLGetData() returns SQL\_SUCCESS and the size of the last data chunk.

### GranteeList CLI/ODBC configuration keyword

Reduces the amount of information returned when the application gets a list of table or column privileges.

**db2cli.ini keyword syntax:**

```
GranteeList = " 'userID1', 'userID2',... 'userIDn' "
```

**Default setting:**

Do not filter the results.

**Usage notes:**

This option can be used to reduce the amount of information returned when the application gets a list of privileges for tables in a database, or columns in a table. The list of authorization IDs specified is used as a filter; the only tables or columns that are returned are those with privileges that have been granted *TO* those IDs.

Set this option to a list of one or more authorization IDs that have been granted privileges, delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
GranteeList=" 'USER1', 'USER2', 'USER8' "
```

In the above example, if the application gets a list of privileges for a specific table, only those columns that have a privilege granted *TO* USER1, USER2, or USER8 would be returned.

---

### GrantorList CLI/ODBC configuration keyword

Reduces the amount of information returned when the application gets a list of table or column privileges.

**db2cli.ini keyword syntax:**

```
GrantorList = " 'userID1', 'userID2',... 'userIDn' "
```

**Default setting:**

Do not filter the results.

**Usage notes:**

This option can be used to reduce the amount of information returned when the application gets a list of privileges for tables in a database, or columns in a table. The list of authorization IDs specified is used as a filter; the only tables or columns that are returned are those with privileges that have been granted *BY* those IDs.

Set this option to a list of one or more authorization IDs that have granted privileges, delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
GrantorList=" 'USER1', 'USER2', 'USER8' "
```

In the above example, if the application gets a list of privileges for a specific table, only those columns that have a privilege granted *BY* USER1, USER2, or USER8 would be returned.

---

## Graphic CLI/ODBC configuration keyword

Specifies if CLI returns SQL\_GRAPHIC (double-byte character) as a supported SQL data type and what unit is used to report GRAPHIC column length.

**db2cli.ini keyword syntax:**

Graphic = 0 | 1 | 2 | 3

**Default setting:**

The SQL\_GRAPHIC data type is not returned as a supported SQL data type, and the length of GRAPHIC columns equals the maximum number of DBCS characters in the column.

**Usage Notes:**

The Graphic keyword controls whether the SQL\_GRAPHIC (double-byte character) data type is reported as a supported SQL data type when SQLGetTypeInfo() is called, as well as what unit is used to report the length of GRAPHIC columns for all CLI functions that return length or precision as either output arguments or as part of a result set.

Set the Graphic keyword as follows:

- 0 - SQL\_GRAPHIC is not returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of DBCS characters in the column.
- 1 - SQL\_GRAPHIC is returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of DBCS characters in the column.
- 2 - SQL\_GRAPHIC is not returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of bytes in the column.
- 3 - SQL\_GRAPHIC is returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of bytes in the column.

---

## Hostname CLI/ODBC configuration keyword

Specifies the server system's host name or IP address, used with file DSN or in a DSN-less connection.

**db2cli.ini keyword syntax:**

Hostname = *host name* | *IP Address*

**Default setting:**

None

**Only applicable when:**

Protocol set to TCPIP

**Usage notes:**

Use this option in conjunction with the ServiceName option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the Protocol option is set to TCPIP.

## IgnoreWarnList CLI/ODBC configuration keyword

Specify either the server system's host name or its IP address.

---

## IgnoreWarnList CLI/ODBC configuration keyword

Ignores specified sqlstates.

**db2cli.ini keyword syntax:**

IgnoreWarnList = "sqlstate1', 'sqlstate2', ..."

**Default setting:**

Warnings are returned as normal

**Usage notes:**

On rare occasions an application may not correctly handle some warning messages, but does not want to ignore all warning messages. This keyword can be used to indicate which warnings are not to be passed on to the application. The IgnoreWarnings keyword should be used if all database manager warnings are to be ignored.

If an sqlstate is included in both IgnoreWarnList and WarningList, it will be ignored altogether.

Each sqlstate must be in uppercase, delimited with single quotes and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
IgnoreWarnList="'01000', '01004', '01504'"
```

---

## IgnoreWarnings CLI/ODBC configuration keyword

Ignores database manager warnings.

**db2cli.ini keyword syntax:**

IgnoreWarnings = 0 | 1

**Default setting:**

Warnings are returned as normal.

**Usage notes:**

On rare occasions, an application will not correctly handle warning messages. This keyword can be used to indicate that warnings from the database manager are not to be passed to the application. The possible settings are:

- 0 - Warnings are reported as usual (default)
- 1 - Database manager warnings are ignored and SQL\_SUCCESS is returned. Warnings from the DB2 CLI/ODBC driver are still returned; many are required for normal operation.

Although this keyword can be used on its own, it can also be used with the WarningList CLI/ODBC configuration keyword.

---

## Instance CLI/ODBC configuration keyword

Specifies the instance name for a local IPC connection for file DSN or DSN-less connectivity.

**db2cli.ini keyword syntax:**

Instance = *instance name*



### Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

When you set this keyword, you must also set the following options:

- Database
- Protocol=IPC

---

## Interrupt CLI/ODBC configuration keyword

Sets the interrupt processing mode.

### db2cli.ini keyword syntax:

Interrupt = 0 | 1 | 2

### Default setting:

1

### Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- Database
- Protocol=IPC

The keyword values have the following meaning:

- 0 Disables interrupt processing (SQLCancel calls will not interrupt the processing.)
- 1 Interrupts are supported (default.) In this mode, if the server supports an interrupt, an interrupt will be sent. Otherwise the connection is dropped.

The settings for `INTERRUPT_ENABLED` (a DB2 Connect gateway setting) and the DB2 registry variable `DB2CONNECT_DISCONNECT_ON_INTERRUPT` will take precedence over the Interrupt keyword setting of 1.

- 2 Interrupt drops the connection regardless of server's interrupt capabilities (SQLCancel will drop the connection.)

---

## KRBPlugin CLI/ODBC configuration keyword

Specifies the name of the Kerberos plug-in library to be used for client side authentication for file DSN or DSN-less connectivity.

### db2cli.ini keyword syntax:

KRBPlugin = *plugin name*

### Default setting:

By default, the value is null on UNIX operating systems, and IBMkrb5 on Windows operating systems.

### Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the name of the Kerberos plug-in library to be used for client-side connection authentication. The plug-in is used when the client is authenticated using KERBEROS authentication.

### KeepDynamic CLI/ODBC configuration keyword

Specifies if KEEP\_DYNAMIC functionality is available to CLI applications.

**db2cli.ini keyword syntax:**

KeepDynamic = 0 | 1

**Default setting:**

KEEP\_DYNAMIC functionality is not available to CLI applications.

**Equivalent connection attribute:**

SQL\_ATTR\_KEEP\_DYNAMIC

**Usage notes:**

The KeepDynamic CLI/ODBC configuration keyword should be set according to how the CLI packages were bound on the DB2 for z/OS and OS/390 server. Set KeepDynamic as follows:

- 0 - if the CLI packages on the server were bound with the KEEP\_DYNAMIC NO option
- 1 - if the CLI packages on the server were bound with the KEEP\_DYNAMIC YES option

It is recommended that when KeepDynamic is used, the CurrentPackageSet CLI/ODBC keyword also be set. Refer to the documentation about enabling KEEP\_DYNAMIC support for details on how these keywords can be used together.

---

### LOBCacheSize CLI/ODBC configuration keyword

Specifies maximum cache size (in bytes) for LOBs.

**db2cli.ini keyword syntax:**

LOBCacheSize = *positive integer*

**Default setting:**

LOBs are not cached.

**Equivalent connection or statement attribute:**

SQL\_ATTR\_LOB\_CACHE\_SIZE

**Usage notes:**

The use of LOB locators when retrieving unbound LOB data can be avoided by setting this keyword. For example, if an application does not bind a column prior to calling SQLFetch() and then calls SQLGetData() to fetch the LOB, if LOBCacheSize was set to a value large enough to contain the entire LOB being fetched, then the LOB is retrieved from the LOB cache rather than from a LOB locator. Using the LOB cache instead of the LOB locator in this case improves performance.

Servers that support Dynamic Data Format, also known as progressive streaming, optimize the return of LOB and XML data depending on the actual length of the data. The LOB and XML data can be returned in its entirety, or as an internal token called a progressive reference. CLI manages progressive reference data retrieval. The LobCacheSize defaults to 1MB if a progressive reference is possible for LOB data. Dynamic Data Format progressive references are always used even if LOBCacheSize is not set explicitly. It will be used for any actual LOB instance that exceeds 1MB by default.

## LOBCacheSize CLI/ODBC configuration keyword

For applications that are querying data on a server that supports Dynamic Data Format, setting the LOBCacheSize keyword sets a threshold that is used to determine if the data is returned in its entirety, or as a progressive reference. If the data has a length greater than the LOBCacheSize threshold value, the progressive reference will be returned to CLI to manage, but if the data has a length less than or equal to the LOBCacheSize threshold value, the data will be returned in its entirety.

For applications that are querying data on a server that does not support Dynamic Data Format, the LOBCacheSize threshold value specifies the maximum defined size of a LOB that CLI will buffer in memory. If the defined size of a LOB exceeds the value LOBCacheSize is set to, then the LOB will not be cached. For example, consider a table that is created with a CLOB column of 100MB currently holding 20MB of data, with LOBCacheSize set to 50MB. In this case, even though the size of the LOB itself (20MB) is less than the value set through LOBCacheSize, the CLOB column will not be cached because the defined CLOB size (100MB) exceeds the maximum cache size set through LOBCacheSize (50MB).

ClientBuffersUnboundLOBS is a related keyword.

---

## LOBFileThreshold CLI/ODBC configuration keyword

Specifies the maximum number of bytes of LOB data buffered when SQLPutData() is used.

**db2cli.ini keyword syntax:**

LOBFileThreshold = *positive integer*

**Default setting:**

25 MB

**Usage notes:**

This option specifies the maximum number of bytes of LOB data that CLI will buffer in memory on calls to SQLPutData(). If the specified cache size is exceeded, a temporary file will be created on disk to hold the LOB data before it is sent to the server.

---

## LOBMaxColumnSize CLI/ODBC configuration keyword

Overrides the default value in the COLUMN\_SIZE column for LOB data types.

**db2cli.ini keyword syntax:**

LOBMaxColumnSize = *integer greater than zero*

**Default setting:**

2 Gigabytes (1G for DBCLOB)

**Only applicable when:**

LongDataCompat or MapXMLDescribe with a LOB type is used.

**Usage notes:**

This will override the 2 Gigabyte (1G for DBCLOB) value that is returned by SQLGetTypeInfo() for the COLUMN\_SIZE column for SQL\_CLOB, SQL\_BLOB, and SQL\_DBCLOB and SQL\_XML SQL data types. For SQL\_XML, LOBMaxColumnSize

## LOBMaxColumnSize CLI/ODBC configuration keyword

must be specified with MapXMLDescribe set to a LOB type. Subsequent CREATE TABLE statements that contain LOB columns will use the column size value you set here instead of the default.

---

## LoadXAInterceptor CLI/ODBC configuration keyword

Loads the XA Interceptor for debugging.

**db2cli.ini keyword syntax:**

LoadXAInterceptor = 0 | 1

**Default setting:**

The XA Interceptor is not loaded.

**Usage notes:**

This keyword loads the XA Interceptor for debugging purposes in MTS.

---

## LockTimeout CLI/ODBC configuration keyword

Sets the default value of the LOCKTIMEOUT configuration parameter.

**db2cli.ini keyword syntax:**

LockTimeout = -1 | 0 | **positive integer** ≤ 32767

**Default setting:**

Timeout is turned off (-1), with the application waiting for a lock until either the lock is granted or deadlock occurs.

**Usage notes:**

The LockTimeout keyword specifies the number of seconds a CLI application will wait to obtain locks. If the keyword is set to 0, locks will not be waited for. The -1 setting causes the application to wait indefinitely until either the lock is granted or deadlock occurs.

---

## LongDataCompat CLI/ODBC configuration keyword

Reports LOBs as long data types or as large object types.

**db2cli.ini keyword syntax:**

LongDataCompat = 0 | 1 | 2

**Default setting:**

Reference LOB data types as large object types.

**Equivalent connection attribute:**

SQL\_ATTR\_LONGDATA\_COMPAT

**Usage notes:**

This option indicates to CLI what data type the application expects when working with a database with large object (LOB) columns.

The values for this option are:

- 0 = Reference LOB data types as large object types.
- 1 = Report LOBs as long data types for CLI/ODBC applications only.

## LongDataCompat CLI/ODBC configuration keyword

- 2 = Report LOBs as long data types for JDBC applications only. This does not affect applications using the DB2 Universal JDBC Driver.

Table 160. Corresponding large object and long data types for LOB data

Database data type	Large objects (0 - Default)	Long data types (1 — CLI/ODBC; 2 — JDBC)
CLOB	SQL_CLOB	SQL_LONGVARCHAR
BLOB	SQL_BLOB	SQL_LONGVARBINARY
DBCLOB	SQL_DBCLOB	SQL_LONGVARGRAPHIC*

\* If the MapGraphicDescribe keyword is set in conjunction with LongDataCompat, DBCLOB columns will return an SQL type of SQL\_LONGVARCHAR if MapGraphicDescribe is 1 and SQL\_WLONGVARCHAR if MapGraphicDescribe is 2.

This option is useful when running ODBC applications that cannot handle the large object data types.

The CLI/ODBC option LOBMaxColumnSize can be used in conjunction with this option to reduce the default size declared for the data.

---

## MapBigintCDefault CLI/ODBC configuration keyword

Specifies the default C type of BIGINT columns and parameter markers.

**db2cli.ini keyword syntax:**

MapBigintCDefault = 0 | 1 | 2

**Default setting:**

The default C type representation for BIGINT data is SQL\_C\_BIGINT.

**Usage notes:**

MapBigintCDefault controls the C type that is used when SQL\_C\_DEFAULT is specified for BIGINT columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which cannot handle 8-byte integers. Set MapBigintCDefault as follows:

- 0 - for the default SQL\_C\_BIGINT C type representation
- 1 - for an SQL\_C\_CHAR C type representation
- 2 - for an SQL\_C\_WCHAR C type representation

This keyword affects the behavior of CLI functions where SQL\_C\_DEFAULT might be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData()

---

## MapCharToWChar CLI/ODBC configuration keyword

Specifies the default SQL type associated with SQL\_CHAR, SQL\_VARCHAR, SQL\_LONGVARCHAR.

**db2cli.ini keyword syntax:**

MapCharToWChar = 0 | 1

**Default setting:**

The default SQL type representation for SQL\_CHAR, SQL\_VARCHAR and SQL\_LONGVARCHAR is used.

**Equivalent connection attribute:**

SQL\_ATTR\_MAPCHAR

## MapCharToWChar CLI/ODBC configuration keyword

### Usage notes:

MapCharToWChar controls the SQL type that is returned when describing SQL\_CHAR, SQL\_VARCHAR and SQL\_LONGVARCHAR columns or parameter markers.

Set MapCharToWChar as follows:

- 0 - to return the default SQL type representation
- 1 - to return SQL\_CHAR as SQL\_WCHAR, SQL\_VARCHAR as SQL\_WVARCHAR, and SQL\_LONGVARCHAR as SQL\_WLONGVARCHAR

Only the following CLI functions are affected by setting MapCharToWChar:

- SQLColumns()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()

---

## MapDateCDefault CLI/ODBC configuration keyword

Specifies the default C type of DATE columns and parameter markers.

### db2cli.ini keyword syntax:

MapDateCDefault = 0 | 1 | 2

### Default setting:

The default C type representation for DATE data is SQL\_C\_TYPE\_DATE.

### Usage notes:

MapDateCDefault controls the C type that is used when SQL\_C\_DEFAULT is specified for DATE columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which assume SQL\_C\_CHAR as the default C type for datetime values. Set MapDateCDefault as follows:

- 0 - for the default SQL\_C\_TYPE\_DATE C type representation: a struct containing numeric members for year, month and day
- 1 - for an SQL\_C\_CHAR C type representation: "2004-01-01"
- 2 - for an SQL\_C\_WCHAR C type representation: "2004-01-01" in UTF-16.

This keyword affects the behavior of CLI functions where SQL\_C\_DEFAULT may be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

---

## MapDateDescribe CLI/ODBC configuration keyword

Controls the SQL data type returned when DATE columns and parameter markers are described.

### db2cli.ini keyword syntax:

MapDateDescribe = 0 | 1 | 2

## MapDateDescribe CLI/ODBC configuration keyword

### Default setting:

The default SQL data type for DATE data is returned: SQL\_DATE for ODBC 2.0 or SQL\_TYPE\_DATE for ODBC 3.0.

### Usage notes:

To control the SQL data type that is returned when DATE columns and parameter markers are described, set MapDateDescribe as follows:

- 0 - to return the default SQL data type: SQL\_DATE for ODBC 2.0 or SQL\_TYPE\_DATE for ODBC 3.0
- 1 - to return the SQL\_CHAR SQL data type
- 2 - to return the SQL\_WCHAR SQL data type

Only the following CLI functions are affected by setting MapDateDescribe:

- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

---

## MapDecimalFloatDescribe CLI/ODBC configuration keyword

Specifies the default C type and reported data type of DECFLOAT columns and parameter markers.

### db2cli.ini keyword syntax:

**MapDecimalFloatDescribe** = 0 | 1 | 2 | 3

### Default setting:

0

### Usage notes:

**MapDecimalFloatDescribe** controls the default C type to be used for columns and parameters with a data type of DECFLOAT. It affects the behavior of CLI functions for which SQL\_C\_DEFAULT can be specified as the C type of a column or parameter. Examples of such functions include SQLBindParameter(), SQLBindCol(), and SQLGetData().

**MapDecimalFloatDescribe** also controls the type that will be reported for columns and parameters that have a data type of DECFLOAT. This affects CLI functions that return information about parameters and columns. Examples of such functions include SQLColAttribute() and SQLDescribeParam().

Use this configuration keyword for applications that cannot handle decimal float types or when you would rather always deal with decimal float types as some other type.

Here are the allowed values:

## MapDecimalFloatDescribe CLI/ODBC configuration keyword

Table 161. Valid values for MapDecimalFloatDescribe

Value	DECFLOAT columns and parameters are reported as being this type	DECFLOAT columns and parameters use this default C type
0	SQL_DECFLOAT	SQL_C_CHAR
1	SQL_VARCHAR	SQL_C_CHAR
2	SQL_WVARCHAR	SQL_C_WCHAR
3	SQL_DOUBLE	SQL_C_DOUBLE

---

## MapGraphicDescribe CLI/ODBC configuration keyword

Controls the SQL data type returned when GRAPHIC, VARGRAPHIC, and LONGVARGRAPHIC columns and parameter markers are described.

**db2cli.ini keyword syntax:**

MapGraphicDescribe = 0 | 1 | 2

**Default setting:**

The default SQL data types are returned: SQL\_GRAPHIC for GRAPHIC columns, SQL\_VARGRAPHIC for VARGRAPHIC columns, and SQL\_LONGVARGRAPHIC for LONG VARGRAPHIC columns.

**Usage notes:**

To control the SQL data type that is returned when GRAPHIC-based columns and parameter markers are described, set MapGraphicDescribe as follows:

- 0 - to return the default SQL data types
- 1 - to return the CHAR-based SQL data types: SQL\_CHAR for GRAPHIC columns, SQL\_VARCHAR for VARGRAPHIC columns, and SQL\_LONGVARCHAR for LONG VARGRAPHIC columns
- 2 - to return the WCHAR-based SQL data types: SQL\_WCHAR for GRAPHIC columns, SQL\_WVARCHAR for VARGRAPHIC columns, and SQL\_WLONGVARCHAR for LONG VARGRAPHIC columns

Only the following CLI functions are affected by setting MapGraphicDescribe:

- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

---

## MapTimeCDefault CLI/ODBC configuration keyword

Specifies the default C type of TIME columns and parameter markers.

**db2cli.ini keyword syntax:**

MapTimeCDefault = 0 | 1 | 2

**Default setting:**

The default C type representation for TIME data is SQL\_C\_TYPE\_TIME.



## MapTimeCDefault CLI/ODBC configuration keyword

### Usage notes:

MapTimeCDefault controls the C type that is used when SQL\_C\_DEFAULT is specified for TIME columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which assume SQL\_C\_CHAR as the default C type for datetime values. Set MapTimeCDefault as follows:

- 0 - for the default SQL\_C\_TYPE\_TIME C type representation: a struct containing numeric members for hour, minute, and second
- 1 - for an SQL\_C\_CHAR C type representation: "12:34:56"
- 2 - for an SQL\_C\_WCHAR C type representation: "12:34:56" in UTF-16.

This keyword affects the behavior of CLI functions where SQL\_C\_DEFAULT may be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

**Note:** MapTimeCDefault supersedes Patch2=24. If both MapTimeCDefault and Patch2=24 are set, the MapTimeCDefault value takes precedence.

---

## MapTimeDescribe CLI/ODBC configuration keyword

Controls the SQL data type returned when TIME columns and parameter markers are described.

### db2cli.ini keyword syntax:

MapTimeDescribe = 0 | 1 | 2

### Default setting:

The default SQL data type for TIME data is returned: SQL\_TIME for ODBC 2.0 or SQL\_TYPE\_TIME for ODBC 3.0

### Usage notes:

To control the SQL data type that is returned when TIME columns and parameter markers are described, set MapTimeDescribe as follows:

- 0 - to return the default SQL data type: SQL\_TIME for ODBC 2.0 or SQL\_TYPE\_TIME for ODBC 3.0
- 1 - to return the SQL\_CHAR SQL data type
- 2 - to return the SQL\_WCHAR SQL data type

Only the following CLI functions are affected by setting MapTimeDescribe:

- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

---

## MapTimestampCDefault CLI/ODBC configuration keyword

Specifies the default C type of TIMESTAMP columns and parameter markers.

## MapTimestampCDefault CLI/ODBC configuration keyword

### db2cli.ini keyword syntax:

MapTimestampCDefault = 0 | 1 | 2

### Default setting:

The default C type representation for TIMESTAMP data is SQL\_C\_TYPE\_TIMESTAMP.

### Usage notes:

MapTimestampCDefault controls the C type that is used when SQL\_C\_DEFAULT is specified for TIMESTAMP columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which assume SQL\_C\_CHAR as the default C type for datetime values. Set MapTimestampCDefault as follows:

- 0 - for the default SQL\_C\_TYPE\_TIMESTAMP C type representation: a struct containing numeric members for year, month, day, hour, minute, second, and fraction of a second
- 1 - for an SQL\_C\_CHAR C type representation: "2004-01-01 12:34:56.123...*n*", where *n*=12.
- 2 - for an SQL\_C\_WCHAR C type representation: "2004-01-01 12:34:56.123456...*n*" in UTF-16, where *n*=12.

This keyword affects the behavior of CLI functions where SQL\_C\_DEFAULT may be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

---

## MapTimestampDescribe CLI/ODBC configuration keyword

Controls the SQL data type returned when TIMESTAMP columns and parameter markers are described.

### db2cli.ini keyword syntax:

MapTimestampDescribe = 0 | 1 | 2

### Default setting:

The default SQL data type for TIMESTAMP data is returned: SQL\_TIMESTAMP for ODBC 2.0 or SQL\_TYPE\_TIMESTAMP for ODBC 3.0.

### Usage notes:

To control the SQL data type that is returned when TIMESTAMP columns and parameter markers are described, set MapTimestampDescribe as follows:

- 0 - to return the default SQL data type: SQL\_TIMESTAMP for ODBC 2.0 or SQL\_TYPE\_TIMESTAMP for ODBC 3.0
- 1 - to return the SQL\_CHAR SQL data type
- 2 - to return the SQL\_WCHAR SQL data type

Only the following CLI functions are affected by setting MapTimeStamPDescribe:

- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()

## MapTimestampDescribe CLI/ODBC configuration keyword

- SQLProcedureColumns()
- SQLSpecialColumns()

---

## MapXMLCDefault CLI/ODBC configuration keyword

Controls the default C type representation used when SQL\_C\_DEFAULT is specified for XML columns and parameter markers.

**db2cli.ini keyword syntax:**

MapXMLCDefault = 0 | 1 | 2 | 3

**Default setting:**

The default C type representation for XML data is SQL\_C\_BINARY.

**Usage notes:**

MapXMLCDefault controls the C type that is used when SQL\_C\_DEFAULT is specified for XML columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which might assume SQL\_C\_WCHAR as the default C type for XML values. Set MapXMLCDefault as follows:

- 0 - for the default SQL\_C\_BINARY C type representation
- 1 - for the SQL\_C\_CHAR C type representation; this can result in data loss as the XML data is converted to the local application code page
- 2 - for the SQL\_C\_WCHAR C type representation

This keyword affects the behaviour of CLI functions where SQL\_C\_DEFAULT can be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

---

## MapXMLDescribe CLI/ODBC configuration keyword

Controls the SQL data type returned when XML columns and parameter markers are described.

**db2cli.ini keyword syntax:**

MapXMLDescribe = -370 | -350 | -152 | -99 | -98

**Default setting:**

The default SQL data type for XML data is returned: SQL\_XML (-370)

**Usage notes:**

To control the SQL data type that is returned when XML columns and parameter markers are described, set MapXMLDescribe to one of the following integer values:

- -370 to return the default SQL\_XML SQL data type
- -350 to return the SQL\_DBCLOB SQL data type
- -152 to return the SQL\_SS\_XML SQL data type

**Note:** The SQL\_SS\_XML value of -152 belongs to the reserved range of Microsoft SQL Server and is not defined by IBM.

- -99 to return the SQL\_BLOB SQL data type
- -98 to return the SQL\_CLOB SQL data type

## MapXMLDescribe CLI/ODBC configuration keyword

The data length for XML values mapped to LOB types is the maximum length for the mapped data type.

When used in conjunction with the LongDataCompat keyword set to the value 1, XML values mapped to LOB data types will be mapped to the corresponding LONG data type as well.

Character types specified for MapXMLDescribe may result in data loss during data conversion if the application code page does not support all of the characters in the source data. Mapping XML values to character types, therefore, is only recommended with caution.

This keyword is recommended to provide compatibility with applications that access XML columns as CLOB or BLOB, or use Microsoft application development technologies.

---

## MaxLOBBlockSize CLI/ODBC configuration keyword

Specifies the maximum return block size for LOB or XML data.

**db2cli.ini keyword syntax:**

MaxLOBBlockSize = 0 | ... | 2147483647

**Default setting:**

There is no limit to the data block size for LOB or XML data.

**Equivalent connection or statement attribute:**

SQL\_ATTR\_MAX\_LOB\_BLOCK\_SIZE

**Usage notes:**

During data retrieval, the server will include all of the information for the current row in its reply to the client even if the maximum block size has been reached.

If both MaxLOBBlockSize and the db2set registry variable DB2\_MAX\_LOB\_BLOCK\_SIZE are specified, the value for MaxLOBBlockSize will be used.

---

## Mode CLI/ODBC configuration keyword

Sets the default connection mode.

**db2cli.ini keyword syntax:**

Mode = SHARE | EXCLUSIVE

**Default setting:**

SHARE

**Not applicable when:**

connecting to a host or IBM Power Systems™ server.

**Usage notes:**

Sets the CONNECT mode to either SHARE or EXCLUSIVE. If a mode is set by the application at connect time, this value is ignored. The default is SHARE.

---

## NotifyLevel CLI/ODBC configuration keyword

Sets the diagnostic level.

### db2cli.ini keyword syntax:

NotifyLevel = 0 | 1 | 2 | 3 | 4

### Default setting:

3

### Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is equivalent to the database manager parameter NOTIFYLEVEL.

---

## OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword

Makes OLE DB flag LONG data types with DBCOLUMNFLAGS\_ISLONG.

### db2cli.ini keyword syntax:

OleDbReportIsLongForLongTypes = 0 | 1

### Equivalent connection attribute:

SQL\_ATTR\_REPORT\_ISLONG\_FOR\_LONGTYPES\_OLEDB

### Default setting:

LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) do not have the DBCOLUMNFLAGS\_ISLONG flag set, which might cause the columns to be used in the WHERE clause.

### Usage notes:

The OLE DB client cursor engine and the OLE DB .NET Data Provider CommandBuilder object generate UPDATE and DELETE statements based on column information provided by the IBM DB2 OLE DB Provider. If the generated statement contains a LONG type in the WHERE clause, the statement will fail because LONG types cannot be used in a search with an equality operator. Setting the keyword OleDbReportIsLongForLongTypes to 1 will make the IBM DB2 OLE DB Provider report LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) with the DBCOLUMNFLAGS\_ISLONG flag set. This will prevent the long columns from being used in the WHERE clause.

The OleDbReportIsLongForLongTypes keyword is supported by the following database servers:

- DB2 for z/OS
  - version 6 with PTF UQ93891
  - version 7 with PTF UQ93889
  - version 8 with PTF UQ93890
  - versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
  - version 8.2 (equivalent to Version 8.1, FixPak 7) and later

---

## OleDbReturnCharAsWChar CLI/ODBC configuration keyword

Controls how the IBM DB2 OLE DB Provider describes CHAR, VARCHAR, LONG VARCHAR, and CLOB data.

### db2cli.ini keyword syntax:

OleDbReturnCharAsWChar = 0 | 1

## OleDbReturnCharAsWChar CLI/ODBC configuration keyword

### Default setting:

The IBM DB2 OLE DB Provider describes CHAR, VARCHAR, LONG VARCHAR, and CLOB data as DBTYPE\_WSTR.

### Usage notes:

The IBM DB2 OLE DB Provider describes CHAR, VARCHAR, LONG VARCHAR, and CLOB data as DBTYPE\_WSTR by default as of DB2 luw Version 8.1.2. The CLI/ODBC configuration keyword OleDbReturnCharAsWChar allows you to change this default to have the previously stated character data types reported as DBTYPE\_STR.

The available settings are:

- 0 - CHAR, VARCHAR, LONG VARCHAR, and CLOB data are described as DBTYPE\_STR, and the code page of data in ISequentialStream is the local code page of the client
- 1 - CHAR, VARCHAR, LONG VARCHAR, and CLOB data are reported as DBTYPE\_WSTR, and the code page of data in ISequentialStream is UCS-2

---

## OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword

Makes OLE DB's IDBSchemaRowset::GetRowset(DBSCHEMA\_COLUMNS) return a row set sorted by the ORDINAL\_POSITION column.

### db2cli.ini keyword syntax:

OleDbSQLColumnsSortByOrdinal = 0 | 1

### Equivalent connection attribute:

SQL\_ATTR\_SQLCOLUMNS\_SORT\_BY\_ORDINAL\_OLEDB

### Default setting:

IDBSchemaRowset::GetRowset(DBSCHEMA\_COLUMNS) returns the row set sorted by the columns TABLE\_CATALOG, TABLE\_SCHEMA, TABLE\_NAME, COLUMN\_NAME.

### Usage notes:

The Microsoft OLE DB specification requires that IDBSchemaRowset::GetRowset(DBSCHEMA\_COLUMNS) returns the row set sorted by the columns TABLE\_CATALOG, TABLE\_SCHEMA, TABLE\_NAME, COLUMN\_NAME. The IBM DB2 OLE DB Provider conforms to the specification. However, applications that use the Microsoft ODBC Bridge provider (MSDASQL) have been typically coded to get the row set sorted by ORDINAL\_POSITION. Setting the OleDbSQLColumnsSortByOrdinal keyword to 1 will make the provider return a row set sorted by ORDINAL\_POSITION.

The OleDbSQLColumnsSortByOrdinal keyword is supported by the following database servers:

- DB2 for z/OS
  - version 6 with PTF UQ93891
  - version 7 with PTF UQ93889
  - version 8 with PTF UQ93890
  - versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
  - version 8.2 (equivalent to Version 8.1, FixPak 7) and later

## OnlyUseBigPackages CLI/ODBC configuration keyword

Allows the CLI applications to use only big packages.

**db2cli.ini keyword syntax:**

**OnlyUseBigPackages=** 0 | 1

**Default setting:**

CLI applications use small packages (3 by default) with 64 sections, and once these sections are exhausted, it starts using big packages, with 384 sections.

**Usage notes:**

This option can be used to allow the applications to access to a large number of sections (384) in big packages, as opposed to 64 sections in small packages.

When the **OnlyUseBigPackages=1** keyword is set, the small CLI packages should not be used.

---

## OptimizeForNRows CLI/ODBC configuration keyword

Appends 'OPTIMIZE FOR n ROWS' clause to every select statement.

**db2cli.ini keyword syntax:**

**OptimizeForNRows =** *integer*

**Default setting:**

The clause is not appended.

**Equivalent statement attribute:**

SQL\_ATTR\_OPTIMIZE\_FOR\_NROWS

**Usage notes:**

This option will append the "OPTIMIZE FOR n ROWS" clause to every select statement, where n is an integer larger than 0. If set to 0 (the default) this clause will not be appended.

---

## PWD CLI/ODBC configuration keyword

Defines the default password.

**db2cli.ini keyword syntax:**

**PWD =** *password*

**Default setting:**

None

**Usage notes:**

Use this *password* if a password is not provided by the application at connect time.

The password is stored as plain text in the db2cli.ini file and is therefore not secure.

## PWD CLI/ODBC configuration keyword

In Version 9.7 Fix Pack 5 and later fix packs, a password phrase can be used as the password when accessing DB2 for z/OS servers. A password phrase is a character string consisting of mixed-case letters, numbers, and special characters including blanks. Password phrases have security advantages over passwords in that they are long enough to withstand most hacking attempts yet are unlikely to be written down because they are so easy to remember.

---

## PWDPlugin CLI/ODBC configuration keyword

Specifies the name of the userid-password plug-in library to be used for client side authentication for file DSN or DSN-less connectivity.

**db2cli.ini keyword syntax:**

PWDPlugin = *plugin name*

**Default setting:**

By default, the value is null and the DB2-supplied userid-password plug-in library is used.

**Usage notes:**

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the name of the userid-password plug-in library to be used for client-side connection authentication. The plug-in is used when the client is authenticated using `SERVER` or `SERVER_ENCRYPT` authentication.

---

## Patch1 CLI/ODBC configuration keyword

Specifies a work-around for known CLI/ODBC application problems.

**db2cli.ini keyword syntax:**

Patch1 = { 0 | 1 | 2 | 4 | 8 | 16 | ... }

**Default setting:**

Use no work-arounds.

**Usage notes:**

This keyword is used to specify a work-around for known problems with ODBC applications. The value specified can be for none, one, or multiple work-arounds. The patch values specified here are used in conjunction with any Patch2 values that might also be set.

Using the CLI/ODBC Settings notebook you can select one or more patches to use. If you set the values in the `db2cli.ini` file itself and want to use multiple patch values then simply add the values together to form the keyword value. For example, if you want the patches 1, 4, and 8, then specify `Patch1=13`.

- 0 = No work around (default)



## Patch1 CLI/ODBC configuration keyword

Table 162. Patch1 CLI/ODBC configuration keyword values

Value	Description
1	<p>Replaces COUNT(<i>exp</i>) with COUNT(*) before sending the SQL statement to the database server.</p> <p>DB2 for z/OS Version 7 and Version 8.1 in "compatibility mode" do not support the syntax COUNT(<i>exp</i>). With PATCH1 = 1 specified, applications that use this syntax can run without modification. Depending on the complexity of the query that is being replaced, this replacement might not produce the expected results.</p> <p>This setting is ignored for DB2 Database for Linux, UNIX, and Windows, and for versions of DB2 for z/OS later than Version 8.1.</p>
4	<p>Changes input timestamp data to date data if the time and fraction part of the timestamp are zero. For example, {ts 'YYYY-MM-DD 00:00:00'} is changed to {d 'YYYY-MM-DD'}. This value is typically needed for older versions of Microsoft Access.</p>
8	<p>Changes input timestamp data to time data if the date part of the timestamp is either 1899-12-30 or 1900-01-01. For example, {ts '1899-12-30 HH:MM:SS'} is changed to {t 'HH:MM:SS'}. This value is typically needed for older versions of Microsoft Access.</p>
64	<p>Null-terminates output GRAPHIC strings. This value is typically needed by Microsoft Access in a double-byte (DBCS) environment.</p>

## Patch1 CLI/ODBC configuration keyword

Table 162. Patch1 CLI/ODBC configuration keyword values (continued)

Value	Description
128	<p>Disables the default performance optimization behavior for the MSysConf table associated with some Microsoft applications.</p> <p>Microsoft applications, such as Microsoft Access, use a configuration table called MSysConf. Once these applications successfully connect to a database, they will typically issue the following query: "SELECT Config, nValue FROM MSysConf". Because the MSysConf table does not exist in a DB2 database by default, this query fails with the error "SQL0204N "MSysConf" is an undefined name.". Microsoft applications can handle this error and continue processing, however, issuing the query across the network to the DB2 server incurs overhead.</p> <p>To enhance performance, CLI assumes that this query will always fail, so when it detects that an application is trying to execute this query, it automatically returns an error with an SQLSTATE of S0002 (Table not found). The query, therefore, is never sent to the server. If, however, the user has created the MSysConf configuration table in the database and wants the application to access it, this PATCH1 value can be set to disable the performance optimization and allow the query to be executed.</p>
256	Service use only
512	Service use only
1024	Returns SQL_SUCCESS_WITH_INFO instead of SQL_NO_DATA_FOUND from the SQLExecute() and SQLExecDirect() functions if the executed UPDATE or DELETE statement affected no rows. This value might be needed by some Microsoft Visual Basic applications.
4096	Prevents a COMMIT from being issued after closing a cursor in autocommit mode.
8192	Returns an extra result set after invoking a stored procedure. This extra result set has one row and consists of the output values of the stored procedure. This PATCH1 value might be needed by some Powerbuilder applications that require an extra result set.
32768	Forces the driver to make Microsoft Query applications work with DB2 for z/OS synonyms.
65536	Deprecated
131072	Deprecated
262144	Deprecated

## Patch1 CLI/ODBC configuration keyword

Table 162. Patch1 CLI/ODBC configuration keyword values (continued)

Value	Description
524288	Deprecated
1048576	Service use only
2097152	Service use only

---

## Patch2 CLI/ODBC configuration keyword

Specifies a workaround for known CLI and ODBC application problems.

### db2cli.ini keyword syntax:

**Patch2** = "patch value 1, patch value 2, patch value 3, ..."

### Default setting:

Use no workarounds

### Usage notes:

You can specify this value for zero, one, or multiple workarounds. You can use the patch values that are specified here with any **Patch1** values that might also be set.

When specifying multiple patches, the values are specified in a comma-delimited string (unlike the **Patch1** option where the values are added together and the sum is used).

- 0 = No work around (default)

To set **Patch2** values 3, 4 and 8 you would specify:

Patch2="3, 4, 8"

Table 163. Patch2 CLI/ODBC configuration keyword values

Value	Description
1	Deprecated
3	Service use only.
4	Deprecated
5	Deprecated
6	Forces CLI to return a message indicating that scrollable cursors are not supported. This setting is required by some applications (such as Visual Basic) that make use of LOBs or that do not require or want scrollable cursors to be used, even though they have been explicitly requested by the application.
7	Maps all GRAPHIC column data types to the CHAR column data type. The precision of a GRAPHIC column will also be doubled; for example, GRAPHIC(20) will be reported as CHAR(40).
8	Ignores catalog search arguments in schema calls.
11	SQLGetInfo() function reports that catalog names are supported for Visual Basic stored procedures.
12	Deprecated

## Patch2 CLI/ODBC configuration keyword

Table 163. Patch2 CLI/ODBC configuration keyword values (continued)

Value	Description
13	Prevents keywords in the db2cli.ini initialization file from being appended to the output connection string.
14	Deprecated
15	Causes a period separator to be used instead of the default locale's decimal separator in character output.
16	Deprecated
17	Deprecated
18	Attempts to replace literals with parameter markers for inefficient applications that use literals repeatedly. It is only applicable to INSERT SQL statements with the VALUES clause using only literals. Coding your application properly to use parameter markers is the best solution.  This value is no longer available in Version 9.7 or later.
19	Removes parentheses from the ON clause of an outer join, where the outer join is an ODBC escape sequence and the server is DB2 for MVS/ESA Version 5. DB2 for MVS/ESA Version 5 does not currently support the ODBC syntax where parentheses are permitted in the ON clause of an outer join clause. Setting this <b>Patch2</b> value allows the outer join escape sequence to be used against DB2 for MVS/ESA Version 5. You should only set this value when the server is DB2 for MVS/ESA Version 5.
20	Forces CLI to rewrite the BETWEEN predicate when the server is DB2 for MVS/ESA. DB2 for MVS/ESA does not currently support the BETWEEN predicate with parameter markers as both operands. Setting this <b>Patch2</b> value causes (expression ? BETWEEN ?) to be rewritten as (expression >= ? and expression <= ?).
21	Deprecated
22	Causes the SQLGetInfo() function to report SQL_OUTER_JOINS=NO and SQL_OJ_CAPABILITIES=0. This prevents the application from using outer joins where they are not supported, thus ensuring that the outer join queries do not fail.
23	Deprecated
24	Reports TIME data as SQL_CHAR data. This patch value is used as a workaround for Microsoft Access applications.

## Patch2 CLI/ODBC configuration keyword

Table 163. Patch2 CLI/ODBC configuration keyword values (continued)

Value	Description
25	Removes trailing zeros in the CHAR representation of DECIMAL columns; used as a workaround for Microsoft Access applications.
28	Deprecated
29	Removes leading zeros in the string representation of DECIMAL values x, where $1 > x > -1$ ; used as a workaround for ADO applications with some MDAC versions.
30	Disables stored procedure caching optimization.
31	Deprecated
32	Deprecated
33	Returns the ISO version of timestamp data when converted to CHAR, rather than the ODBC version.
34	Deprecated
38	Turns statement caching off.
42	Prevents the FOR UPDATE clause from being used with keyset cursors. By default, most applications behaves as though keyset cursors is updatable. However, if updatable cursor is not required, then this <b>Patch2</b> value makes the cursor read-only (but still scrollable and sensitive to changes made by others).
50	Frees LOB locators when the SQLFetch() function is executed, rather than when a COMMIT is issued. This <b>Patch2</b> value frees the locators that are used internally when applications fetch LOB data without binding the LOB columns with the SQLBindCol() function(or equivalent descriptor APIs). Locators that are explicitly returned to the application must still be freed by the application. You can use this <b>Patch2</b> value to avoid scenarios where an application receives SQLCODE = -429 (no more locators).
56	Allows client support for Early Close Cursors for those servers that do not support it as in the case of DB2 Universal Database™ for z/OS and OS/390 version 7 or earlier.
57	Allows calling a stored procedure that returns a NULL output parameter value without providing an output indicator pointer. This is normally applicable to older versions of Borland Delphi products.

## Patch2 CLI/ODBC configuration keyword

Table 163. Patch2 CLI/ODBC configuration keyword values (continued)

Value	Description
58	DateTime values inserted into the database that cause truncation errors can be downgraded to a truncation warning using this <b>Patch2</b> value.
61	When data is given to the client from an SQL_CHAR data type, there might be right padded spaces. This patch value strips off right padded single byte spaces, but not double byte spaces. This behavior partially mimics the Neon Shadow Driver behavior
66	Allows applications to retrieve the regional setting that affects decimal separators in a Windows environment. The regional setting is normally ignored by default. This patch value is ignored if <b>Patch2=15</b> or <b>db2set</b> registry variables <b>DB2TERRITORY</b> or <b>DB2CODEPAGE</b> are set. The only supported decimal separators are the period and comma.
78	Alters the behavior of the SQLGetPosition() function when the source LOB value is in a DBCLOB column on DB2 Universal Database for z/OS and OS/390 Version 7.1 or later. Setting this <b>PATCH2</b> value causes the SQLGetPosition() function to query SYSIBM.SYSDUMMYU instead of SYSIBM.SYSDUMMY1.
81	According to the CLI specification, IBM Data Server Driver for ODBC and CLI should return the column ordinal number for an expression. By default, CLI returns the column ordinal number. According to the ODBC Specification, an ODBC driver should return an empty string as the column names for an expression. If the <b>Patch2</b> CLI/ODBC configuration keyword is set to 81, an empty string is returned for the column name of an expression by the IBM Data Server Driver for ODBC and CLI.
82	Forces the CLI to use the meaning of the SQL_ATTR_REPLACE_QUOTED_LITERALS value instead of the SQL_ATTR_RESET_CONNECTION value . SQL_ATTR_RESET_CONNECTION is supported starting with DB2 Version 9.7 Fix Pack 5 and later Fix Packs

## Port CLI/ODBC configuration keyword

Specifies the server system's service name or port number, used with a file DSN or in a DSN-less connection.

**db2cli.ini** keyword syntax:

Port = *service name* | *port number*

**Default setting:**

None

**Only applicable when:**

Protocol set to TCPIP

**Usage notes:**

Use this option in conjunction with the Hostname option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the Protocol option is set to TCPIP.

Specify either the server system's service name or its port number. The service name must be available for lookup at the client machine if it is used.

**ProgramID CLI/ODBC configuration keyword**

Sets the user-defined character string, that associates an application with a connection. When you set the **ProgramID** keyword, DB2 for z/OS or IBM Informix® database server associates this keyword with any statements that are inserted into the dynamic SQL statement cache.

**db2cli.ini keyword syntax:**

ProgramID = &lt;string&gt;

**Default setting:**

None

**Equivalent connection attribute:**

SQL\_ATTR\_INFO\_PROGRAMID

**Usage notes:**

**Restriction:** This keyword is supported only for CLI applications accessing DB2 Universal Database for z/OS Version 8 and later, IBM Informix database server, or DB2 for i servers.

When monitoring a CLI application, you can use the **ProgramID** keyword to identify a statement that was inserted into the dynamic SQL statement cache. You can use the **ProgramID** keyword to specify an identifier, as a string of up to 80 bytes for connection to DB2 for z/OS, or IBM Informix database server server at the connection and statement level. In IBM DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the **ProgramID** keyword on DB2 for i V6R1 and later servers. DB2 for i servers support up to a length of 255 characters. If you set the **ProgramID** keyword, the SQLGetConnectAttr() and SQLGetStatementAttr() APIs return the specified **ProgramID** keyword value.

**ProgramName CLI/ODBC configuration keyword**

Sets the default client application name to a user-defined name which is used to identify the application at the server when monitoring.

**db2cli.ini keyword syntax:**

ProgramName = &lt;string&gt; | PID

**Default setting:**

The application is identified by the client. By default, the first 20 bytes of the executable name is used.

## ProgramName CLI/ODBC configuration keyword

### Equivalent connection attribute:

SQL\_ATTR\_INFO\_PROGRAMNAME

### Usage notes:

When monitoring a CLI application, it may be useful to identify the application by a user-defined string, instead of by the default identifier that DB2 assigns. ProgramName allows the user to specify the identifier as either a string up to 20 bytes in length or the string "PID" (without the quotation marks).

If ProgramName is set to "PID" for a CLI application, the application's name will consist of the prefix "CLI" along with the application's process ID and the current active connection handle, as follows: CLI<pid>:<connectionHandle#>. The "PID" setting is useful when monitoring application servers that run multiple applications with numerous connections to the same database.

(When the ProgramName keyword is set to "PID" for other types of applications, the "CLI" prefix is replaced with the following values corresponding to the type of application: "JDBC" for JDBC applications, "OLEDB" for OLE DB applications, and "ADONET" for .NET applications.)

---

## PromoteLONGVARtoLOB CLI/ODBC configuration keyword

Promotes LONGVARxxx data types to xLOB data types.

### db2cli.ini keyword syntax:

PROMOTELONGVARTOLOB = 0 | 1

### Default setting:

0

### Usage notes:

This option should only be used when a LONGVARxxxx value has the potential to exceed 32K. The return type of the column is promoted to xLOB to hold the larger data size.

---

## Protocol CLI/ODBC configuration keyword

Communications protocol used for File DSN or in a DSN-less connection.

### db2cli.ini keyword syntax:

Protocol = TCPIP | TCPIP6 | TCPIP4 | IPC | LOCAL

### Default setting:

none

### Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

TCP/IP is the only protocol supported when using a File DSN. Set the option to the string TCPIP (without the slash).

When this option is set then the following options must also be set:

- Database;
- ServiceName; and
- Hostname.



IPC connectivity can be specified by setting Protocol to either **IPC** or **LOCAL**.

When Protocol = **IPC** | **LOCAL** the Instance keyword must also be set.

---

## QueryTimeoutInterval CLI/ODBC configuration keyword

Delay (in seconds) between checks for a query timeout.

### db2cli.ini Keyword Syntax:

QueryTimeoutInterval = 0 | 5 | positive integer

### Default Setting:

5 seconds

### Usage Notes<sup>®</sup>:

An application can use the `SQLSetStmtAttr()` function to set the **SQL\_ATTR\_QUERY\_TIMEOUT** statement attribute. This attribute indicates the number of seconds to wait for an SQL statement or XQuery expression to complete executing before attempting to cancel the execution and returning to the application.

The **QueryTimeoutInterval** configuration keyword is used to indicate how long the CLI driver should wait between checks to see if the query has completed.

For instance, suppose **SQL\_ATTR\_QUERY\_TIMEOUT** is set to 25 seconds (timeout after waiting for 25 seconds), and **QueryTimeoutInterval** is set to 10 seconds (check the query every 10 seconds). The query will not time out until 30 seconds (the first check AFTER the 25 second limit). CLI implements query timeout by starting a thread that periodically queries the status of each executing query. The **QueryTimeoutInterval** value specifies how long the query timeout thread waits between checks for expired queries. Because this is an asynchronous operation to the queries being executed, it is possible that a given query may not be timed out until **SQL\_ATTR\_QUERY\_TIMEOUT** + **QueryTimeoutInterval** seconds. In the example above, the best-case timeout would be at 26 seconds, and the worst-case timeout would be at 35 seconds.

There may be cases where the **SQL\_ATTR\_QUERY\_TIMEOUT** is set to a value which is too low, and the query should NOT be timed-out. If the application cannot be modified (that is, a third party ODBC application), then the **QueryTimeoutInterval** can be set to 0, and the CLI driver will ignore the **SQL\_ATTR\_QUERY\_TIMEOUT** setting, and therefore wait for SQL statements to complete execution before returning to the application.

If **QueryTimeoutInterval** is set to 0, any attempt by the application to set **SQL\_ATTR\_QUERY\_TIMEOUT** will result in SQLSTATE 01S02 (Option Value Changed).

This option is only valid in the **COMMON** section of a initialization file and therefore applies to all connections to DB2 database(s).

Alternatively, **QueryTimeoutInterval** can be set to a value that is larger than the **SQL\_ATTR\_QUERY\_TIMEOUT** setting, thus preventing timeouts from occurring at the specified interval. For example, if the application sets a 15 second **SQL\_ATTR\_QUERY\_TIMEOUT** value, but the server requires at least 30 seconds to execute the query, the **QueryTimeoutInterval** can be set to a value of 30 seconds or so to prevent this query from timing out after 15 seconds.

## QueryTimeoutInterval CLI/ODBC configuration keyword

**Note:**

- The .NET Framework does not support settings in `db2cli.ini` file.
- This CLI keyword is ignored if it is used inside a stored procedure or routine that uses CLI API calls.
- The **QueryTimeoutInterval** can also interrupt a LOAD, which returns SQL3005N instead of SQL0952N.

---

## ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword

Allows a NULL connect to process the COMMON section of the `db2cli.ini` initialization file.

**db2cli.ini keyword syntax:**

`ReadCommonSectionOnNullConnect = 0 | 1`

**Default setting:**

A NULL connect does not process the `db2cli.ini` initialization file.

**Usage notes:**

For use with DB2 CLI, DB2 JDBC Type 2 and DB2 .NET stored procedures, specify 1 to allow stored procedures to read the COMMON section of the `db2cli.ini` file, thus allowing stored procedures to use keywords listed in that section.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

---

## ReceiveTimeout CLI/ODBC configuration keyword

Specifies the time in seconds to wait for a reply from the server on an established connection before terminating the attempt and generating a communication timeout error.

**db2cli.ini keyword syntax:**

`ReceiveTimeout = 0 | 1 | 2 | ... | 32767`

**Default setting:**

The client waits indefinitely for a reply from the server on an established connection.

**Equivalent connection attribute:**

`SQL_ATTR_RECEIVE_TIMEOUT`

**Usage notes:**

This keyword has no effect during connection establishment and is only supported for TCP/IP protocol.

---

## Reopt CLI/ODBC configuration keyword

Enables query optimization or reoptimization of SQL statements that have special registers, global variables, or parameter markers.

**db2cli.ini keyword syntax:**

`Reopt = 2 | 3 | 4`

**Default setting:**

No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for special registers, global variables, or parameter markers.

### Equivalent connection or statement attribute:

SQL\_ATTR\_REOPT

### Usage notes:

Optimization occurs by using the values available at query execution time for the special registers, global variables, or parameter markers instead of the default estimates that are chosen by the compiler. The valid values of the keyword are:

- 2 = SQL\_REOPT\_NONE. This is the default. No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for the special registers, global variables, or parameter markers. The default NULLID package set is used to execute dynamic SQL statements.
- 3 = SQL\_REOPT\_ONCE. Query optimization occurs once at query execution time, when the query is executed for the first time. The NULLIDR1 package set, which is bound with the REOPT ONCE bind option, is used.
- 4 = SQL\_REOPT\_ALWAYS. Query optimization or reoptimization occurs at query execution time every time the query is executed. The NULLIDRA package set, which is bound with the REOPT ALWAYS bind option, is used.

The NULLIDR1 and NULLIDRA are reserved package set names, and when used, REOPT ONCE and REOPT ALWAYS are implied respectively. These package sets have to be explicitly created with the following commands:

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

If both the Reopt and CurrentPackageSet keywords are specified, CurrentPackageSet takes precedence.

---

## ReportPublicPrivileges CLI/ODBC configuration keyword

Reports PUBLIC privileges in SQLColumnPrivileges() and SQLTablePrivileges() results.

### db2cli.ini keyword syntax:

ReportPublicPrivileges = 0 | 1

### Default setting:

PUBLIC privileges are not reported.

### Usage notes:

This keyword specifies if privileges assigned to the PUBLIC group are to be reported as if PUBLIC was a user in the SQLColumnPrivileges() and SQLTablePrivileges() results.

---

## ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword

Returns errors that were uncovered during CLI error recovery as warnings.

### db2cli.ini keyword syntax:

ReportRetryErrorsAsWarnings = 0 | 1

## ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword

**Only applicable when:**

RetryOnError keyword is set to 1.

**Default setting:**

Do not return errors uncovered during CLI error recovery to the application.

**Usage notes:**

By default, when the CLI retry logic is able to recover successfully from a non-fatal error, it masks that error from the application by returning SQL\_SUCCESS. Because application binding errors can be hidden this way, for debugging purposes, you may want to set ReportRetryErrorsAsWarnings to 1. This setting keeps the error recovery on, but forces CLI to return to the application, any errors that were uncovered as warnings.

---

## RetCatalogAsCurrServer CLI/ODBC configuration keyword

Specifies whether catalog functions return the CURRENT SERVER value or a null value for the catalog columns.

**db2cli.ini keyword syntax:**

RetCatalogAsCurrServer= 0 | 1

**Default setting:**

If the target DBMS returns null for the catalog columns, the CURRENT SERVER value will not be substituted.

**Usage notes:**

If the catalog functions for the target DBMS return a null value for the catalog columns, setting RetCatalogAsCurrServer to 1 causes the DBMS to return the CURRENT SERVER value instead.

- 0 = Catalog functions return the null value for the catalog columns (default).
- 1 = Catalog functions return the CURRENT SERVER value, instead of the null value, for the catalog columns.

For example, assume the catalog function SQLTables() returns a result set where the values in the TABLE\_CAT column are null values. Setting RetCatalogAsCurrServer to 1 causes the DBMS to return the CURRENT SERVER value in the TABLE\_CAT column.

---

## RetOleDbConnStr CLI/ODBC configuration keyword

Specifies whether the Mode CLI/ODBC configuration keyword returns a numeric value or string value.

**db2cli.ini keyword syntax:**

RetOleDbConnStr = 0 | 1

**Default setting:**

The value for the Mode CLI/ODBC configuration keyword is returned as a string.

**Usage notes:**

The Mode CLI/ODBC configuration keyword sets the CONNECT mode to either SHARE or EXCLUSIVE. OLE DB expects the value for Mode to have

## RetOleDbConnStr CLI/ODBC configuration keyword

a numeric representation instead of a string representation.  
RetOleDbConnStr toggles between returning a string and a numeric value.

The possible settings are as follows:

- 0 — the value returned by `SQLDriverConnect()` and `SQLBrowseConnect()` for the Mode keyword is either `SHARE` or `EXCLUSIVE`
- 1 — the value returned by `SQLDriverConnect()` and `SQLBrowseConnect()` for the Mode keyword is either 3 (for `SHARE`) or 12 (for `EXCLUSIVE`)

For example, if you set `RetOleDbConnStr=1` and call `SQLDriverConnect()` or `SQLBrowseConnect()` with the following input connection string for a shared connection:

```
DSN=SAMPLE;MODE=SHARE
```

then the output connection string will have the following format:

```
DSN=SAMPLE;UID=;PWD=;MODE=3
```

If you set `RetOleDbConnStr=1` and call `SQLDriverConnect()` or `SQLBrowseConnect()` with the following input connection string for an exclusive connection:

```
DSN=SAMPLE;UID=NEWTON;PWD=SECRET;MODE=EXCLUSIVE
```

then the output connection string will have the following format:

```
DSN=SAMPLE;UID=NEWTON;PWD=SECRET;MODE=12
```

OLE DB applications that use the string representation for the value of the Mode keyword returned by `SQLDriverConnect()` and `SQLBrowseConnect()` will receive an error from OLE DB Component Services. OLE DB Component Services returns an error because it expects the keyword Mode to have numeric values. Setting `RetOleDbConnStr` to 1 avoids this behavior, as the value for Mode will then be numeric.

---

## RetryOnError CLI/ODBC configuration keyword

Turns on or off the CLI driver's error recovery behavior.

**db2cli.ini keyword syntax:**

```
RetryOnError = 0 | 1
```

**Default setting:**

Allow the CLI driver to attempt error recovery on non-fatal errors.

**Usage notes:**

By default, CLI will attempt to recover from non-fatal errors, such as incorrect binding of application parameters, by retrieving additional information on the failing SQL statement and then executing the statement again. The additional information retrieved includes input parameter information from the database catalog tables. If CLI is able to recover successfully from the error, by default, it does not report the error to the application. The CLI/ODBC configuration keyword `ReportRetryErrorsAsWarnings` allows you to set whether error recovery warnings are returned to the application or not.

**Important:** Once CLI has successfully completed the error recovery, the application may behave differently, because CLI will use the catalog information gathered

## RetryOnError CLI/ODBC configuration keyword

during the recovery for subsequent executions of that particular SQL statement, rather than the information provided in the original `SQLBindParameter()` function calls. If you do not want this behavior, set `RetryOnError` to 0, forcing CLI not to attempt recovery. You should, however, modify the application to correctly bind statement parameters.

---

## ReturnAliases CLI/ODBC configuration keyword

Controls whether the CLI schema APIs report aliases in the metadata results.

**db2cli.ini keyword syntax:**

**ReturnAliases = 0 | 1**

**Default setting:**

1: By default, aliases will be considered when qualifying rows for metadata procedures.

**Usage notes:**

This keyword specifies whether aliases (or synonyms) should be considered when qualifying rows for metadata procedures. Not considering aliases can provide significant performance benefits by avoided costly joins with the base tables to determine the additional tables that should qualify for a given query.

- 0 : Aliases will not be considered when qualifying rows for metadata procedures (better performance.)
- 1 : Aliases will be considered when qualifying rows for metadata procedures.

The following CLI APIs are affected by this keyword :

- `SQLColumns()`
- `SQLColumnPrivileges()`
- `SQLTables()`
- `SQLTablePrivileges()`
- `SQLStatistics()`
- `SQLSpecialColumns()`
- `SQLForeignKeys()`
- `SQLPrimaryKeys()`

---

## ReturnSynonymSchema CLI/ODBC configuration keyword

Controls whether CLI schema APIs report the schema name for DB2 for z/OS synonyms in the `TABLE_SCHEM` column of the schema procedure result sets.

**db2cli.ini keyword syntax:**

**ReturnSynonymSchema = 0 | 1**

**Default setting:**

1: By default, the creator of the synonym will be returned in the `TABLE_SCHEM` column.

**Usage notes:**

Valid settings:

- 0 : the `TABLE_SCHEM` column of the procedure result set will be NULL.
- 1 : the `TABLE_SCHEM` column of the procedure result set will contain the creator of the synonym.

## ReturnSynonymSchema CLI/ODBC configuration keyword

You cannot access a synonym on a DB2 for z/OS server using a name qualified with a schema. For this reason, the meaning of the TABLE\_SCHEM column of a CLI schema API result set is different, with respect to synonyms, when you are running against a DB2 for z/OS server.

This CLI keyword has no effect when you use CLI schema APIs against a DB2 Database for Linux, UNIX, and Windows server.

The following CLI APIs are affected by this keyword:

- SQLColumns()
- SQLColumnPrivileges()
- SQLTables()
- SQLTablePrivileges()
- SQLStatistics()
- SQLSpecialColumns()
- SQLForeignKeys()
- SQLPrimaryKeys()

You must have the following program temporary fixes (PTFs) on the DB2 for z/OS database server to use this keyword:

Table 164. DB2 for z/OS PTFs for ReturnSynonymSchema

DB2 for z/OS	PTF or APAR numbers
Version 7	UK13643
Version 8	UK13644
Version 9	

---

## SQLOverrideFileName CLI/ODBC configuration keyword

Specifies the location of the override file, which lists CLI statement attribute settings for particular SQL statements.

### db2cli.ini keyword syntax:

SQLOverrideFileName = <absolute or relative path name>

### Default setting:

No override file is used.

### Usage notes:

The **SQLOverrideFileName** keyword specifies the location of the override file to be read by the CLI driver. An override file contains values for CLI statement attributes that apply to particular SQL statements. For example, you can specify the **SQLOverrideFileName** keyword with path and the override file name in the db2cli.ini file.

```
[MyDatabase]
SQLOverrideFileName=C:\temp\myfile.txt
```

In the c:\temp\ path, you would have override file named myfile.txt, containing values for CLI statement attributes that applies to particular SQL statements. The override file would start with a COMMON section ([COMMON]) containing only one keyword called **Stmts**, which tells you how many statements to override. Following sample override file (myfile.txt) has two statements:

## SQLOverrideFileName CLI/ODBC configuration keyword

```
[Common]
Stmts=2
```

```
[1]
StmtIn=SELECT * FROM Employee
StmtAttr=SQL_ATTR_BLOCK_FOR_NROWS=50;SQL_ATTR_OPTIMIZE_FOR_NROWS=1;
```

```
[2]
StmtIn=SELECT * FROM Sales
StmtAttr=SQL_ATTR_MAX_ROWS=25;
```

The number specified by **Stmts** in the [COMMON] section of the override file equals the number of SQL statements contained in the override file.

---

## SaveFile CLI/ODBC configuration keyword

Specifies the file name of a DSN file in which to save the attribute values of the keywords used in making the present, successful connection.

### db2cli.ini keyword syntax:

You can not set this keyword in the db2cli.ini file.

You can specify the value of this keyword in the connection string in SQLDriverConnect like this:

```
SaveFile = file name
```

---

## SchemaList CLI/ODBC configuration keyword

Restricts schemas that are used to query table information.

### db2cli.ini keyword syntax:

```
SchemaList = " 'schema1', 'schema2',... 'schemaN' "
```

### Default setting:

None

### Usage notes:

You can use the **SchemaList** keyword to specify a schema list to reduce the time that it takes for an application to query table information and reduce the number of tables that are listed by the application. Each schema name is case-sensitive. As shown in the following example, you must delimit a schema name with single quotation marks, separate schema names by commas, and enclose the list in double quotation marks:

```
SchemaList="'USER1','USER2','USER3'"
```

For DB2 for z/OS, you can include the CURRENT SQLID option in the list, but without the single quotation marks, as shown in the following example:

```
SchemaList="'USER1',CURRENT SQLID,'USER3'"
```

The maximum length of the string is 256 characters.

You can use this keyword with the **DBName** and **TableType** keywords to further limit the number of tables for which information is returned.

In DB2 Version 9.7 Fix Pack 5 and later fix packs, you can specify the \*ALL or \*USRLIBOption in the list of schema names to resolve unqualified stored procedure calls or to find libraries in catalog API calls. Use the \*ALL option to search all schemas in the connected database. The \*ALL option is the



## SchemaList CLI/ODBC configuration keyword

default for CLI. For DB2 for i servers, if you specify the \*USRLIBL option, CLI searches the current libraries of the server job. If you are migrating from the IBM i Access ODBC drive, and you specified the \*USRLIBL option for the DBQ or DefaultLibraries connection string keyword, add the \*USRLIBL option to the list of schema names for the **SchemaList** keyword, as shown in the following example:

```
[DSNSAMP]
SCHEMALIST="*USRLIBL"
```

In DB2 Version 9.7 Fix Pack 5, if you specify the \*USRLIBL option for the **SchemaFilter** keyword and specify the \*USRLIBL option along with schema names for the **SchemaList** keyword, a CLI call to the SQLTables() function in DB2 for i returns a result set that contains only tables with schemas belonging to the user library list (\*USRLIBL).

In DB2 Version 9.7 Fix Pack 6 and later fix packs, a CLI call to the following catalog functions in DB2 for i returns a result set that contains database objects with schema that belong to user library list (\*USRLIBL) and schema names listed in the **SchemaList** keyword:

- SQLTables
- SQLTablePrivileges
- SQLColumns
- SQLColumnPrivileges

To obtain database objects with combined user library list (\*USRLIBL) and schema names that you specified for the **SchemaList** keyword, the DB2 for i server must meet the following requirements:

- You must apply PTF Group SF99601 Version 21 or later on DB2 for i V6R1.
- You must apply PTF Group SF99701 Version 11 or later on DB2 for i V7R1.

---

## security CLI/ODBC configuration keyword

Specifies whether or not SSL support is used for File DSN or in a DSN-less connection.

**db2cli.ini keyword syntax:**

**security** = SSL

**Default setting:**

None.

**Usage notes:**

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies whether a TCP/IP communication will be with SSL support or not. It can only be used with the protocols TCPIP, TCPIP4, or TCPIP6. If not specified, a normal TCP/IP without SSL support will be used.

---

## ServerMsgMask CLI/ODBC configuration keyword

Specifies when CLI should request the error message from the server.

## ServerMsgMask CLI/ODBC configuration keyword

### db2cli.ini keyword syntax:

ServerMsgMask = 0 | 1 | -2 | -1

### Default setting:

CLI will check the local message files first to see if the message can be retrieved. If no matching SQLCODE is found, CLI will request the information from the server.

### Equivalent connection attribute:

SQL\_ATTR\_SERVER\_MSGTXT\_MASK

### Usage notes:

This keyword is used in conjunction with the “UseServerMsgSP CLI/ODBC configuration keyword” on page 423. The keyword can be set to:

- **0 (default)** = SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_LOCAL\_FIRST. CLI will check the local message files first to see if the message can be retrieved. If no matching SQLCODE is found, CLI will request the information from the server.
- **1** = SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_WARNINGS. CLI always requests the message information from the server for warnings but error messages are retrieved from the local message files.
- **-2** = SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_ERRORS. CLI always requests the message information from the server for errors but warning messages are retrieved from the local message files.
- **-1** = SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_ALL. CLI always requests the message information from the server for both error and warning messages.

---

## ServiceName CLI/ODBC configuration keyword

The server system's service name or port number, used with file DSN or in a DSN-less connection.

### db2cli.ini keyword syntax:

ServiceName = *service name* | *port number*

### Default setting:

None

### Only applicable when:

Protocol set to TCPIP

### Usage notes:

Use this option in conjunction with the Hostname option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the PROTOCOL option is set to TCPIP.

Specify either the server system's service name or its port number. The service name must be available for lookup at the client machine if it is used.

---

## SkipTrace CLI/ODBC configuration keyword

Excludes CLI trace information from the DB2 trace.

### db2cli.ini keyword syntax:

SkipTrace = 0 | 1

### Default setting:

Do not skip the trace function.

### Usage notes:

This keyword can improve performance by allowing the DB2 trace function to bypass CLI applications. Therefore, if the DB2 trace facility db2trc is turned on and this keyword is set to 1, the trace will not contain information from the execution of the CLI application.

Turning SkipTrace on is recommended for production environments on the UNIX platform where trace information is not required. Test environments may benefit, however, from having trace output, so this keyword can be turned off (or left at its Default setting) when detailed execution information is desired.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## SQLCODEMAP CLI/ODBC configuration keyword

Specifies whether SQLCODE mapping is used or turned off.

### db2cli.ini keyword syntax:

SQLCODEMAP = <MAP> | <NOMAP>

### Default setting:

MAP

### Usage notes:

You can set this keyword in the [Data Source] section of the db2cli.ini file, or in the connection string.

If the value of the keyword is MAP, SQLCODE mapping is used. If you specify the NOMAP option, SQLCODE mapping is turned off.

---

## SSLClientLabel CLI/ODBC configuration keyword

Specifies a unique SSL label that is mapped to a specific certificate to use with CERTIFICATE authentication.

### db2cli.ini keyword syntax:

SSLClientLabel = <label>

### Default setting:

None.

### Usage notes:

The **SSLClientLabel** keyword is only available for use with CERTIFICATE authentication starting in DB2 Version 9.7 Fix Pack 6 and later. You can set this keyword in the [Data Source] section of the db2cli.ini file, or in the connection string.

When you configure certificate-based authentication to supply authentication information, you must specify the **SSLClientLabel** keyword in the db2dsdriver.cfg configuration file or in the db2cli.ini configuration file.

## SSLClientLabel CLI/ODBC configuration keyword

If you set the authentication parameter to the **CERTIFICATE** option but do not specify the **SSLClientLabel** keyword in the `db2cli.ini` configuration file, the `db2dsdriver.cfg` configuration file, or the connection string, error CLI0221E is returned. If you do not set the authentication parameter to the **CERTIFICATE** option but specify the **SSLClientLabel** keyword in the `db2cli.ini` configuration file, the `db2dsdriver.cfg` configuration file, or the connection string, error CLI0222E is returned.

---

## SSLClientKeystash CLI/ODBC configuration keyword

Specifies the SSL stash file used for File DSN or in a DSN-less connection.

**db2cli.ini** keyword syntax:

**SSLClientKeystash** = *<fully qualified stash file path>*

**Default setting:**

None.

**Usage notes:**

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the fully qualified path of the stash file (.sth), which stores an encrypted password to the key database file. The stash file is used to access the key database file during the SSL handshake. This parameter must be defined if the SSL protocol (**security=SSL**) is specified.

The **SSLClientKeystash** keyword is mutually exclusive with the **SSLClientKeystoreDBPassword** keyword. If the SSL protocol (**security=SSL**) is specified, either **SSLClientKeystash** or **SSLClientKeystoreDBPassword** must be specified in the connection string, CLI configuration file, `db2cli.ini`, or in your data server driver configuration file, `db2dsdriver.cfg`. Otherwise, the connection fail error will be returned.

**Note:**

- The **ssl\_client\_keystash** keyword is also supported to provide compatibility with earlier version
- **SSLClientKeystash** keyword is supported starting from DB2 Version 9.7 Fix Pack 6

---

## SSLClientKeystoredb CLI/ODBC configuration keyword

Specifies the SSL key database file that is used for File DSN or in a DSN-less connection.

**db2cli.ini** keyword syntax:

**SSLClientKeystoredb** = *<fully qualified key file path>*

**Default setting:**

None.

**Usage notes:**

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the fully qualified path of the key database file (.kdb). The key database file stores the signer certificate from the server personal certificate.

- For a self-signed server personal certificate, the signer certificate is the public key of the personal certificate.

## SSLClientKeystoredb CLI/ODBC configuration keyword

- For a certificate authority signed server personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.

If the SSL protocol (**security=SSL**) is used, this parameter must be defined. The signer certificate from the server's personal certificate must also exist on the client for authentication to take place.

### Note:

- The **ssl\_client\_keystoredb** keyword is also supported to provide compatibility with earlier version
- **SSLClientKeystoredb** keyword is supported starting from DB2 Version 9.7 Fix Pack 6

---

## SSLClientKeystoreDBPassword CLI/ODBC configuration keyword

Specifies the password for the SSL connection when the authentication parameter is set to CERTIFICATE.

### db2cli.ini keyword syntax:

**SSLClientKeystoreDBPassword** = <password>

### Default setting:

None.

### Usage notes:

The **SSLClientKeystoreDBPassword** keyword is only available for use with CERTIFICATE authentication starting in DB2 Version 9.7 Fix Pack 6 and later. You can set this keyword in the [Data Source] section of the db2cli.ini file, or in the connection string.

The **SSLClientKeystash** and **SSLClientKeystoreDBPassword** configuration parameters are mutually exclusive. If you specify both the **SSLClientKeystash** configuration parameter and the **SSLClientKeystoreDBPassword** configuration parameter in either the db2cli.ini configuration file or the db2dsdriver.cfg configuration file, error CLI0219E is returned.

---

## StaticCapFile CLI/ODBC configuration keyword

Specifies the Capture File name and optionally the path where it will be saved.

### db2cli.ini keyword syntax:

**StaticCapFile** = < Full file name >

### Default setting:

None - you must specify a capture file.

### Only applicable when:

StaticMode is set to Capture or Match

### Usage notes:

This keyword is used to specify the Capture File name and optionally the directory where it will be saved.

### StaticLogFile CLI/ODBC configuration keyword

Specifies the Static Profiling Log File name and optionally the directory where it will be saved.

**db2cli.ini keyword syntax:**

StaticLogFile = < Full file name >

**Default setting:**

No Static Profiling Log is created. If a filename is specified without a pathname then the current path will be used.

**Only applicable when:**

StaticMode is set to Capture or Match

**Usage notes:**

This keyword is used to specify the Static Profiling Log File name and optionally the directory where it will be saved.

---

### StaticMode CLI/ODBC configuration keyword

Specifies whether the CLI/ODBC application will capture SQL or use a static SQL Package for this DSN.

**db2cli.ini keyword syntax:**

StaticMode = DISABLED | CAPTURE | MATCH

**Default setting:**

Disabled - SQL statements are not captured and no static SQL package is used.

**Usage notes:**

This option allows you to specify how the SQL issued by the CLI/ODBC application for this DSN will be processed:

- **DISABLED** = Static mode disabled. No special processing. The CLI/ODBC statements will be executed as dynamic SQL with no change. This is the default.
  - **CAPTURE** = Capture Mode. Execute the CLI/ODBC statements as dynamic SQL. If the SQL statements are successful, they will be captured into a file (known as the Capture File) to be bound by the DB2CAP command later.
  - **MATCH** = Match mode. Execute the CLI/ODBC statements as static SQL statements if a matching statement is found in the Capture Files specified in StaticPackage. The Capture File must first be bound by the DB2CAP command.
- 

### StaticPackage CLI/ODBC configuration keyword

Specifies the package to be used with the static profiling feature.

**db2cli.ini keyword syntax:**

StaticPackage = *collection\_id.package\_name*

**Default setting:**

None - you must specify a package name.

**Only applicable when:**

STATICMODE is set to CAPTURE

### Usage notes:

This keyword is used to specify the package to be used when the application runs in Match Mode. You first need to use Capture Mode to create the Capture File.

Only the first 7 characters of the indicated package name will be used. A one-byte suffix will be added to represent each isolation level, as follows:

- 0 for Uncommitted Read (UR)
- 1 for Cursor Stability (CS)
- 2 for Read Stability (RS)
- 3 for Repeatable Read (RR)
- 4 for No Commit (NC)

---

## StmtConcentrator CLI/ODBC configuration keyword

Starting in Version 9.7 Fix Pack 3 and later fix packs, DB2 supports this keyword, which specifies whether dynamic statements that contain literal values use the statement cache.

### db2cli.ini keyword syntax:

```
StmtConcentrator = OFF | WITHLITERALS
```

### Default setting:

The default behavior specified for statement concentration on the server.

### Equivalent environment or connection attribute:

```
SQL_ATTR_STMT_CONCENTRATOR
```

### Usage notes:

This option specifies whether dynamic statements that contain literal values use the statement cache.

- OFF - The statement concentrator is disabled.
- WITHLITERALS - The statement concentrator with literal behavior is enabled for situations that are supported by the server. For example, the statement concentrator is not enabled if the statement has parameter markers, named parameter markers, or a mix of literals, parameter markers, and named parameter markers.

Starting in V9.7 Fix Pack 3a, the StmtConcentrator keyword is supported with DB2 for z/OS V10. When you use this attribute against DB2 for z/OS servers older than Version 10, the request is ignored.

---

## StreamGetData CLI/ODBC configuration keyword

Optimizes data output stream for SQLGetData() function.

### db2cli.ini keyword syntax:

```
StreamGetData = 0 | 1
```

### Default setting:

CLI buffers all the data on the client.

### Equivalent connection or statement attribute:

```
SQL_ATTR_STREAM_GETDATA
```

### Usage notes:

The StreamGetData keyword is ignored if Dynamic Data Format, also known as progressive streaming, is not supported by the server. For

## StreamGetData CLI/ODBC configuration keyword

applications that do not need to buffer data and are querying data on a server that supports Dynamic Data Format, specify 1 to indicate that data buffering is not required. The CLI client will optimize the data output stream.

If StreamGetData is set to 1 and CLI cannot determine the number of bytes still available to return in the output buffer, SQLGetData() returns SQL\_NO\_TOTAL (-4) as the length when truncation occurs. Otherwise, SQLGetData() returns the number of bytes still available.

---

## StreamPutData CLI/ODBC configuration keyword

Improves performance for data passed through SQLPutData() function calls on one statement handle, by writing data directly to the internal connection-level communication buffer.

**db2cli.ini keyword syntax:**

StreamPutData = 0 | 1

**Default setting:**

Do not write data directly to the connection-level buffer; write to the default statement-level buffer instead.

**Usage notes:**

By default, CLI writes data passed in through SQLPutData() function calls to an internal statement-level buffer. On the subsequent SQLParamData() call, the contents of the buffer are then written to an internal connection-level communication buffer and sent to the server. If only one statement handle is used to insert data into a target database on a particular connection at a given point in time, then you can improve performance by setting StreamPutData=1. This causes CLI to write the put data directly to the connection-level buffer. If, however, multiple statements concurrently insert data into a target database on a particular connection, then setting StreamPutData=1 may decrease performance and result in unexpected application errors, as the statements in the shared connection-level communication buffer will be prone to serialization.

---

## SysSchema CLI/ODBC Configuration Keyword

Sets an alternative schema to be searched in place of the SYSIBM schema.

**db2cli.ini keyword syntax:**

SysSchema = **alternative schema**

**Default setting:**

The default table qualifier name used when querying DB2 for z/OS is SYSIBM.

**Usage notes:**

This option indicates an alternative schema, or table qualifier, to be searched in place of the SYSIBM schema when the CLI and ODBC Catalog Function calls are issued to obtain system catalog information from DB2 for z/OS.

Using this new schema name, the system administrator can define a set of views, or a copies of the tables, consisting of a subset of the rows for system catalog tables such as:

- SYSIBM.SYSCOLAUTH



- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSDATATYPES
- SYSIBM.SYSFOREIGNKEYS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSKEYS
- SYSIBM.SYSKEYCOLUSES
- SYSIBM.SYSPARMS
- SYSIBM.SYSRELS
- SYSIBM.SYSROUTINES
- SYSIBM.SYSTABAUTH
- SYSIBM.SYSTABCONST
- SYSIBM.SYSTABLES
- SYSIBM.SYSSYNONYMS

For example, if the set of views, or a copies tables, for the system catalog tables is in the ACME schema, then the view (or copy of the table) for SYSIBM.SYSTABLES is ACME.SYSTABLES; and **SysSchema** should be set to ACME.

For applications that automatically query the system catalogs for all table names, defining and using limited views of the system catalog tables reduces the number of tables listed by the application. This can reduce the time it takes for the application to query table information since a subset of table names is returned.

Defining and using copies of the system catalog tables, with the same indexes defined on the copy as those defined on the system table, can reduce the time it takes for applications to query the database.

The **SchemaList**, **TableType** and **DBName** keywords can be used in conjunction with the **SysSchema** keyword to further limit the number of tables for which information is returned.

For more information about which system catalog tables can be used with **SysSchema**, and about the function of **SysSchema**, refer to the documentation for APAR PK05102 by visiting:

Support for IBM mainframes

and searching for "PK05102".

---

### TableType CLI/ODBC configuration keyword

Defines a default list of TABLETYPES returned when querying table information.

**db2cli.ini keyword syntax:**

```
TableType = " 'TABLE' | ',ALIAS' | ',VIEW' | ',INOPERATIVE  
VIEW' | ',SYSTEM TABLE' | ',SYNONYM' "
```

**Default setting:**

No default list of TABLETYPES is defined.

**Usage notes:**

If there is a large number of tables defined in the database, a tabletype string can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application.

## TableType CLI/ODBC configuration keyword

Any number of the values can be specified. Each type must be delimited with single quotes, separated by commas, and in uppercase. The entire string must also be enclosed in double quotes. For example:

```
TableType="'TABLE', 'VIEW'"
```

This option can be used in conjunction with DBNAME and SCHEMALIST to further limit the number of tables for which information will be returned.

TableType is used to provide a default for the CLI function that retrieves the list of tables, views, aliases, and synonyms in the database. If the application does not specify a table type on the function call, and this keyword is not used, information about all table types is returned. If the application does supply a value for the *tabletype* on the function call, then that argument value will override this keyword value.

If TableType includes any value other than TABLE, then the DBName keyword setting cannot be used to restrict information to a particular DB2 for z/OS database.

---

## TargetPrincipal CLI/ODBC configuration keyword

Specifies the fully qualified Kerberos principal name of the DB2 instance owner for a target server.

**db2cli.ini keyword syntax:**

```
TargetPrincipal = name/instance@REALM
```

**Default setting:**

None

**Usage notes:**

For Windows 2000, Windows XP, and Windows Server 2003, the fully qualified Kerberos principal name is the DB2 server service logon account in one of the following forms: *userid@DOMAIN*, *userid@xxx.xxx.xxx.com*, or *domain\userid*. For example, if the DB2 server service account is *LocalSystem* then the TargetPrincipal is *HOST/host\_name@DOMAIN*, where *host\_name* is the fully qualified host name and *DOMAIN* is the fully qualified domain name in uppercase. Otherwise, the TargetPrincipal is *userid@DOMAIN*, where *userid* is the user ID for the DB2 server service account and *DOMAIN* is the fully qualified domain name in uppercase.

You can add this keyword to the db2cli.ini file or the db2dsdriver.cfg file. This keyword can set a data source in the [DATA SOURCE] or [COMMON] section of the db2cli.ini file.

---

## TempDir CLI/ODBC configuration keyword

Defines the directory used for temporary files.

**db2cli.ini keyword syntax:**

```
TempDir = < full path name >
```

**Default setting:**

Use the system temporary directory specified by the TEMP or TMP environment variables.

**Usage notes:**

## TempDir CLI/ODBC configuration keyword

When working with Large Objects (CLOBs, BLOBs, etc...), when data conversion occurs, or when data is sent to the server in pieces, a temporary file is often created on the client machine to store the information. Using this option you can specify a location for these temporary files. The system temporary directory will be used if nothing is specified.

The keyword is placed in the data source specific section of the db2cli.ini file, and has the following syntax:

- TempDir= F:\DB2TEMP

The path specified must already exist and the user executing the application must have the appropriate authorities to write files to it. When the DB2 CLI Driver attempts to create temporary files, an SQLSTATE of HY507 will be returned if the path name is invalid, or if the temporary files cannot be created in the directory specified.

---

## TimestampTruncErrToWarning CLI/ODBC configuration keyword

Sets the return value for an overflow of fractional seconds in TIMESTAMP.

**db2cli.ini keyword syntax:**

TimestampTruncErrToWarning = 0 | 1

**Default setting:**

The overflow of fractional seconds in TIMESTAMP results in an error (SQLSTATE 22007).

**Usage notes:**

TimestampTruncErrToWarning controls whether the overflow of fractional seconds in TIMESTAMP results in an error (SQLSTATE 22007) or a warning (SQLSTATE 01S07).

Set TimestampTruncErrToWarning as follows:

- 0 - to return the default error (SQLSTATE 22007)
- 1 - to return the warning (SQLSTATE 01S07)

---

## Trace CLI/ODBC configuration keyword

Turns on the CLI/ODBC trace facility.

**db2cli.ini keyword syntax:**

Trace = 0 | 1 | db2trc

**Default setting:**

No trace information is captured.

**Equivalent environment attribute:**

SQL\_ATTR\_TRACE

**Usage notes:**

**Note:** Starting in version 9.7 FP3, the CLI trace mechanism is also supported with instance-less clients, (IBM data server drivers).

When this option is set to value (1), CLI/ODBC trace records are appended to the file indicated by the TraceFileName configuration parameter or to files in the

## Trace CLI/ODBC configuration keyword

subdirectory indicated by the TracePathName configuration parameter. Trace CLI/ODBC configuration keyword will have no effect if neither TraceFileName or TracePathname is set.

The following example shows how to setup a CLI/ODBC trace file that is written directly to disk:

```
[COMMON]
Trace=1
TraceFileName=E:\TRACES\CLI\MONDAY.CLI
TraceFlush=1
```

When this option is set to value (db2trc), the trace facility of the DB2 instance or the DB2 Administration Server (DAS) will be started. The trace facility is controlled by the **db2trc** command and records information about operations and formats this information into readable form. Enabling the trace facility (OFF by default) might impact your system's performance. As a result, only use the trace facility when directed by a DB2 technical support representative; otherwise, turn off the trace once enough information has been recorded.

Setting the Trace CLI/ODBC configuration keyword to (db2trc) will automatically run **db2trc on** command with -cli option. To stop tracing, command db2trc off must be run by the user.

To create a CLI/ODBC trace file from the trace records recorded by the trace facility, the trace records must be dumped and formatted from the internal trace buffer. For example, after finishing tracing an application, the following steps should be followed to dump and format the CLI/ODBC trace:

1. db2trc dump <dump\_filename>
2. db2trc off
3. db2trc fmt -cli <dump\_filename> <ODBC-CLI\_trace\_filename>

Setting the Trace CLI/ODBC configuration keyword to value (db2trc) provides better performance than value (1). Also, the dumped CLI Trace file is smaller than the CLI Trace file generated with Trace value (1). Using the Trace CLI/ODBC configuration keyword value (db2trc) starts the **db2trc** trace command with a default in memory trace buffer size. With larger CLI applications, the default trace buffer size may be insufficient to prevent trace records from wrapping. To avoid this limitation, run the command **db2trc on** with the **-f** parameter to trace to a file. Here is an example command for tracing CLI to a file named clitr.c.dmp:

```
db2trc on -f clitr.c.dmp -m *.*.CLITRC.*.*
```

After tracing to a file, the trace records do not need to be dumped, but do still require formatting to obtain a final CLI Trace file.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

---

## TraceAPIList CLI/ODBC configuration keyword

Specifies what APIs to trace using the CLI trace facility. If you do not set the **TraceAPIList** keyword, all APIs are traced.

To set the **TraceAPIList** keyword in the db2cli.ini file, issue the following command:

```
db2 update cli cfg for section common using TRACEAPILIST API ID,API ID...
```

## TraceAPIList CLI/ODBC configuration keyword

where *API ID* is an integer that corresponds to the name of a CLI API. You can find the mapping between API names and IDs in the `/sqllib/include/sqlcli1.h` file, as shown in the following example:

```
#define SQL_API_SQLALLOCHANDLE      1001
#define SQL_API_SQLFREEHANDLE      1006
#define SQL_API_SQLCLOSECURSOR     1003
#define SQL_API_SQLENDTRAN         1005
#define SQL_API_SQLCOLATTRIBUTE     6
#define SQL_API_SQLGETSTMTATTR     1014
#define SQL_API_SQLGETCONNECTATTR  1007
```

The `/sqllib/samples/cli/db2conn.c` sample program depicts the behavior of different CLI connect APIs: `SQLConnect`, `SQLDriverConnect`, and `SQLBrowseConnect`. If you do not set the **TraceAPIList** keyword, the CLI trace that is generated is similar to the following example:

```
[ Process: 4453, Thread: 47717036514016 ]
[ Date & Time:      06/26/2009 03:14:51.158736 ]
[ Product:         QDB2/LINUX8664 DB2 v9.7.0.1 ]
[ Level Identifier: 08020107 ]
[ CLI Driver Version: 09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location: /home/mayprasa/sqllib/cfg/db2cli.ini ]
[ CLI Driver Type:  IBM DB2 Application Runtime Client ]
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
phOutput=&00007fffb229a990 )
```

```
----> Time elapsed - 0 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
```

```
<--- SQL_SUCCESS Time elapsed - +2.830000E-004 seconds
```

```
SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3,
cbParam=0 )
```

```
----> Time elapsed - +4.300000E-005 seconds
```

```
SQLSetEnvAttr( )
```

```
<--- SQL_SUCCESS Time elapsed - +1.800000E-005 seconds
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fffb229a8e4 )
```

```
----> Time elapsed - +3.600000E-005 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
```

```
<--- SQL_SUCCESS Time elapsed - +4.480000E-004 seconds
```

```
SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="", cbUID=-3, szAuthStr="",
cbAuthStr=-3 )
```

```
----> Time elapsed - +2.000000E-005 seconds
```

```
( DBMS NAME="DB2/LINUX8664", Version="09.07.0001", Fixpack="0x28020107" )
```

```
( Application Codepage=819, Database Codepage=1208,
```

```
Database XML Codepage=1208,
```

```
Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=1200,
```

```
XML Send/Recv Codepage=1208 )
```

```
SQLConnect( )
```

```
<--- SQL_SUCCESS Time elapsed - +1.242704E+000 seconds
```

```
( DSN=""SAMPLE"" )
```

```
( UID="" )
```

```
( PWD="" )
```

## TraceAPIList CLI/ODBC configuration keyword

To trace only the SQLAllocHandle API, issue the following command:

```
db2 update cli cfg for section common using TRACEAPILIST 1001
```

where 1001 is the ID for the SQLAllocHandle API. The CLI trace that is generated is similar to the following example:

```
[ Process: 5977, Thread: 47628919556832 ]
[ Date & Time:          06/26/2009 03:17:25.066017 ]
[ Product:             QDB2/LINUX8664 DB2 v9.7.0.1 ]
[ Level Identifier:    08020107 ]
[ CLI Driver Version:  09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location: /home/mayprasa/sqllib/cfg/db2cli.ini ]
[ CLI Driver Type:     IBM DB2 Application Runtime Client ]
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
phOutput=&00007fff3657bc70 )
----> Time elapsed - 0 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
<--- SQL_SUCCESS Time elapsed - +2.720000E-004 seconds
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fff3657bbc4 )
----> Time elapsed - +8.000000E-005 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
<--- SQL_SUCCESS Time elapsed - +4.250000E-004 seconds
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fff3657bbc4 )
----> Time elapsed - +1.303675E+000 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
<--- SQL_SUCCESS Time elapsed - +1.920000E-004 seconds
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fff3657bbc0 )
----> Time elapsed - +1.154550E+000 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
<--- SQL_SUCCESS Time elapsed - +1.320000E-004 seconds
```

### Usage notes:

You must enable CLI TRACE to use the **TraceAPIList** keyword.

---

## TraceAPIList! CLI/ODBC configuration keyword

Specifies what APIs not to trace using the CLI trace facility. If you do not set the **TraceAPIList!** keyword, all APIs are traced.

To set the **TraceAPIList!** keyword in the `thedb2cli.ini` file, issue the following command:

```
"db2 update cli cfg for section common using 'TRACEAPILIST!'API ID,API ID..."
```

where *API ID* is an integer that corresponds to the name of a CLI API. You can find the mapping between API names and IDs in the `/sqllib/include/sqlcli1.h` file, as shown in the following example:

## TraceAPIList! CLI/ODBC configuration keyword

```
#define SQL_API_SQLALLOCHANDLE      1001
#define SQL_API_SQLFREEHANDLE      1006
#define SQL_API_SQLCLOSECURSOR     1003
#define SQL_API_SQLENDTRAN         1005
#define SQL_API_SQLCOLATTRIBUTE     6
#define SQL_API_SQLGETSTMTATTR     1014
#define SQL_API_SQLGETCONNECTATTR  1007
```

**Note:** To run this command, you must use double quotation marks around the full command and single quotation marks around the **TRACEAPILIST!** keyword.

The `/sqllib/samples/cli/db2conn.c` sample program depicts the behavior of different CLI connect APIs: `SQLConnect`, `SQLDriverConnect`, and `SQLBrowseConnect`. If you do not set the **TraceAPIList!** keyword, the CLI trace that is generated is similar to the following example:

```
[ Process: 4453, Thread: 47717036514016 ]
[ Date & Time:      06/26/2009 03:14:51.158736 ]
[ Product:         QDB2/LINUX8664 DB2 v9.7.0.1 ]
[ Level Identifier: 08020107 ]
[ CLI Driver Version: 09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location: /home/mayprasa/sqllib/cfg/db2cli.ini ]
[ CLI Driver Type:  IBM DB2 Application Runtime Client ]
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
phOutput=&00007ffffb229a990 )
----> Time elapsed - 0 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
<--- SQL_SUCCESS Time elapsed - +2.830000E-004 seconds
```

```
SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3, cbParam=0 )
----> Time elapsed - +4.300000E-005 seconds
```

```
SQLSetEnvAttr( )
<--- SQL_SUCCESS Time elapsed - +1.800000E-005 seconds
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007ffffb229a8e4 )
----> Time elapsed - +3.600000E-005 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
<--- SQL_SUCCESS Time elapsed - +4.480000E-004 seconds
```

```
SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="", cbUID=-3,
szAuthStr="",
cbAuthStr=-3 )
----> Time elapsed - +2.000000E-005 seconds
( DBMS NAME="DB2/LINUX8664", Version="09.07.0001", Fixpack="0x28020107" )
( Application Codepage=819, Database Codepage=1208, Database XML Codepage=1208,
Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=1200,
XML Send/Recv Codepage=1208 )
```

```
SQLConnect( )
<--- SQL_SUCCESS Time elapsed - +1.242704E+000 seconds
( DSN=""SAMPLE"" )
( UID="" )
( PWD="" )
```

To prevent the `SQLAllocHandle` API from being traced, issue the following command:

## TraceAPIList! CLI/ODBC configuration keyword

```
"db2 update cli cfg for section common using 'TRACEAPILIST!' 1001"
```

where 1001 is the ID for **SQLA11ocHandle** API. The CLI trace that is generated is similar to the following example:

```
[ Process: 5442, Thread: 47530732661472 ]
[ Date & Time:      06/26/2009 05:11:10.794067 ]
[ Product:         QDB2/LINUX8664 DB2 v9.7.0.1 ]
[ Level Identifier: 08020107 ]
[ CLI Driver Version: 09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location: /home/mayprasa/sql/lib/cfg/db2cli.ini ]
[ CLI Driver Type:   IBM DB2 Application Runtime Client ]

SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3, cbParam=0 )
----> Time elapsed - 0 seconds

SQLSetEnvAttr( )
<--- SQL_SUCCESS   Time elapsed - +2.200000E-005 seconds

SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="", cbUID=-3, szAuthStr="",
cbAuthStr=-3 )
----> Time elapsed - +4.850000E-004 seconds
( DBMS NAME="DB2/LINUX8664", Version="09.07.0001", Fixpack="0x28020107" )
( Application Codepage=819, Database Codepage=1208, Database XML Codepage=1208,
Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=1200,
XML Send/Recv Codepage=1208 )

SQLConnect( )
<--- SQL_SUCCESS   Time elapsed - +1.156099E+000 seconds
( DSN="SAMPLE" )
( UID=" " )
( PWD="" )

SQLDisconnect( hDbc=0:1 )
----> Time elapsed - +4.400000E-005 seconds

SQLDisconnect( )
<--- SQL_SUCCESS   Time elapsed - +1.197430E-001 seconds
```

### Usage notes:

You must enable CLI TRACE to use the **TraceAPIList!** keyword.

---

## TraceComm CLI/ODBC configuration keyword

Specifies whether information about each network request is included in the trace file.

### db2cli.ini keyword syntax:

```
TraceComm = 0 | 1
```

### Default setting:

0 - No network request information is captured.

### Only applicable when:

the CLI/ODBC Trace option is turned on.

### Usage notes:



## TraceComm CLI/ODBC configuration keyword

When TraceComm is set on (1) then the following information about each network request will be included in the trace file:

- which CLI functions are processed completely on the client and which CLI functions involve communication with the server
- the number of bytes sent and received in each communication with the server
- the time spent communicating data between the client and server

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceErrImmediate CLI/ODBC configuration keyword

Specifies whether diagnostic records are written to the CLI/ODBC trace when records are generated.

**db2cli.ini keyword syntax:**

TraceErrImmediate = 0 | 1

**Default setting:**

Diagnostic records are only written to the trace file when SQLGetDiagField() or SQLGetDiagRec() is called; or "Unretrieved Error Message" is written to the trace file for handles which had diagnostic records that were left unretrieved.

**Only applicable when:**

the CLI/ODBC Trace option is turned on.

**Usage notes:**

Setting TraceErrImmediate=1 helps in determining when errors occur during application execution by writing diagnostic records to the CLI/ODBC trace file at the time the records are generated. This is especially useful for applications that do not retrieve diagnostic information using SQLGetDiagField() and SQLGetDiagRec(), because the diagnostic records that were generated on a handle will be lost if they are not retrieved or written to the trace file before the next function is called on the handle.

If TraceErrImmediate=0 (the default setting), then diagnostic records will only be written to the trace file if an application calls SQLGetDiagField() or SQLGetDiagRec() to retrieve diagnostic information. If the application does not retrieve diagnostic information through function calls and this keyword is set to 0, then the "Unretrieved Error Message" entry will be written to the trace file if a diagnostic record exists, when a function is next called on the handle.

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceFileName CLI/ODBC configuration keyword

Specifies a file to which all CLI/ODBC trace information is written.

**db2cli.ini keyword syntax:**

TraceFileName = < fully qualified file name >

## TraceFileName CLI/ODBC configuration keyword

**Default setting:**

None

**Only applicable when:**

the Trace option is turned on.

**Usage notes:**

If the file specified does not exist, then it will be created; otherwise, the new trace information will be appended to the end of the file. However, the path the file is expected in must exist.

If the filename given is invalid or if the file cannot be created or written to, no trace will occur and no error message will be returned.

This option is only used when the Trace option is turned on. This will be done automatically when you set this option in the CLI/ODBC Configuration utility.

The TracePathName option will be ignored if this option is set.

CLI trace should only be used for debugging purposes. It will slow down the execution of the CLI/ODBC driver, and the trace information can grow quite large if it is left on for extended periods of time.

The TraceFileName keyword option should not be used with multi-process or multithreaded applications as the trace output for all threads or processes will be written to the same log file, and the output for each thread or process will be difficult to decipher. Furthermore, semaphores are used to control access to the shared trace file which could change the behavior of multithreaded applications. There is no default DB2 CLI trace output log file name.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

---

## TraceFlush CLI/ODBC configuration keyword

Forces a write to disk after n CLI/ODBC trace entries.

**db2cli.ini keyword syntax:**

TraceFlush = 0 | **positive integer**

**Default setting:**

Do not write after every entry.

**Only applicable when:**

the CLI/ODBC Trace option is turned on.

**Usage notes:**

TraceFlush specifies how often trace information is written to the CLI trace file. By default, TraceFlush is set to 0 and each DB2 CLI trace file is kept open until the traced application or thread terminates normally. If the application terminates abnormally, some trace information that was not written to the trace log file may be lost.

Set this keyword to a positive integer to force the CLI driver to close and re-open the appropriate trace file after the specified number of trace entries. The smaller

## TraceFlush CLI/ODBC configuration keyword

the value of the TraceFlush keyword, the greater the impact CLI tracing has on the performance of the application. Setting TraceFlush=1 has the most impact on performance, but will ensure that each entry is written to disk before the application continues to the next statement.

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceFlushOnError CLI/ODBC configuration keyword

Specifies whether all CLI/ODBC trace entries are written to disk when an error occurs.

**db2cli.ini keyword syntax:**

TraceFlushOnError = 0 | 1

**Default setting:**

Do not write CLI/ODBC trace entries as soon as an error occurs.

**Only applicable when:**

the CLI/ODBC Trace option is turned on.

**Usage notes:**

Setting TraceFlushOnError=1 forces the CLI driver to close and re-open the trace file each time an error is encountered. If TraceFlushOnError is left at its default value of 0, then trace file will only be closed when the application terminates normally or the interval specified by the TraceFlush keyword is reached. If the application process were to terminate abnormally when TraceFlushOnError=0, then valuable trace information may be lost. Setting TraceFlushOnError=1 may impact performance, but will ensure that trace entries associated with errors are written to disk.

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceLocks CLI/ODBC configuration keyword

Only trace lock timeouts in the CLI/ODBC trace.

**db2cli.ini keyword syntax:**

TraceLocks = 0 | 1

**Default setting:**

Trace information is not limited to only lock timeouts.

**Only applicable when:**

the Trace option is turned on.

**Usage notes:**

When TraceLocks is set to 1, lock timeouts will be recorded in the trace file.

This option is only used when the CLI/ODBC TRACE option is turned on.

## TraceLocks CLI/ODBC configuration keyword

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TracePIDList CLI/ODBC configuration keyword

Restricts the process IDs for which the CLI/ODBC trace will be enabled.

### db2cli.ini keyword syntax:

TracePIDList = <no value specified> | <comma-delimited list of process IDs>

### Default setting:

All of the process IDs will be traced when the CLI/ODBC trace is run.

### Usage notes:

Use this keyword for applications that create many processes. Capturing the CLI/ODBC trace for such applications can generate many trace files. By using this keyword you can collect the trace of specific problematic processes of applications.

If no value is specified for this keyword, all process IDs will be traced. Otherwise, specify a comma-delimited list of process IDs which you want to be traced when the CLI/ODBC trace runs.

The TraceRefreshInterval keyword must be set to some value before initializing your application, otherwise, the TracePIDList keyword will not take effect.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

To use the TracePIDList keyword:

1. Ensure the Trace CLI/ODBC keyword is set to zero or is not specified in the db2cli.ini file.
2. Add the TraceRefreshInterval CLI/ODBC keyword to the Common section of the db2cli.ini file as follows:

```
[COMMON]
TraceRefreshInterval=<some positive integer>
```
3. Start your application.
4. Using an operating system command such as **ps** (on a UNIX and Linux-based operating systems), determine the process IDs of the processes that you want to collect the CLI/ODBC trace for.
5. Turn CLI/ODBC tracing on and add the process IDs identified to the Common section of the db2cli.ini file by including the following keywords:

```
[COMMON]
Trace=1
TracePathName=<fully-qualified subdirectory name>
TracePIDList=<comma-delimited list of process IDs>
```

CLI/ODBC traces containing information of the process IDs specified will be located in the directory specified by the TracePathName keyword.. You might also see extra empty files that can be ignored.

---

## TracePIDTID CLI/ODBC configuration keyword

Captures the process ID and thread ID for each item being traced.

## TracePIDTID CLI/ODBC configuration keyword

### db2cli.ini keyword syntax:

TracePIDTID = 0 | 1

### Default setting:

The process ID and thread ID for the trace entries are not captured.

### Only applicable when:

the Trace option is turned on.

### Usage notes:

When TracePIDTID is set to 1, the process ID and thread ID for each captured item will be recorded in the trace file. This effect is helpful when the Trace keyword is enabled and multiple applications are executing. This is because Trace writes trace information for all executing applications to a single file. Enabling TracePIDTID differentiates the recorded information by process and thread.

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TracePathName CLI/ODBC configuration keyword

Specifies the subdirectory to be used to store individual CLI/ODBC trace files.

### db2cli.ini keyword syntax:

TracePathName = < **fully qualified subdirectory name** >

### Default setting:

None

### Only applicable when:

the Trace option is turned on.

### Not applicable when:

the TraceFileName option is turned on.

### Usage notes:

Each thread or process that uses the same DLL or shared library will have a separate CLI/ODBC trace file created in the specified directory. A concatenation of the application process ID and the thread sequence number is automatically used to name trace files.

No trace will occur, and no error message will be returned, if the subdirectory given is invalid or if it cannot be written to.

This option is only used when the Trace option is turned on. This will be done automatically when you set this option in the CLI/ODBC Configuration utility.

It will be ignored if the CLI/ODBC option TraceFileName is used.

CLI trace should only be used for debugging purposes. It will slow down the execution of the CLI/ODBC driver, and the trace information can grow quite large if it is left on for extended periods of time.

## TracePathName CLI/ODBC configuration keyword

If both `TraceFileName` and `TracePathName` are specified, the `TraceFileName` keyword takes precedence and `TracePathName` will be ignored.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceRefreshInterval CLI/ODBC configuration keyword

Sets the interval (in seconds) at which the `Trace` and `TracePIDList` keywords are read from the Common section of the `db2cli.ini` file.

**db2cli.ini keyword syntax:**

`TraceRefreshInterval = 0 | positive integer`

**Default setting:**

The `Trace` and `TracePIDList` keywords will only be read from the `db2cli.ini` file when the application is initialized.

**Usage notes:**

Setting this keyword before an application is initialized allows you to dynamically turn off the CLI/ODBC trace within *n* seconds.

**Note:** Setting `TraceRefreshInterval` while the application is running will have no effect. For this keyword to take effect, it must be set before the application is initialized.

Only the `Trace` and `TracePIDList` keywords will be refreshed from the `db2cli.ini` file if this keyword is set. No other CLI or ODBC configuration keywords will be reread.

When `TraceRefreshInterval` is set to a nonzero positive integer value, a thread is spawned to monitor the `db2cli.ini`. In this situation, the applications connected to the database need to be multithread safe; otherwise the application might behave in an unexpected manner.

This keyword is contained in the Common section of the initialization file and therefore applies to all connections to DB2.

**Note:** This CLI keyword is ignored if it is used inside a stored procedure or routine that uses CLI API calls.

---

## TraceStmtOnly CLI/ODBC configuration keyword

Only trace dynamic SQL statements in the CLI/ODBC trace.

**db2cli.ini keyword syntax:**

`TraceStmtOnly = 0 | 1`

**Default setting:**

Trace information is not limited to only dynamic SQL statements.

**Only applicable when:**

the `Trace` option is turned on.

**Usage notes:**

When `TraceStmtOnly` is set to 1, only dynamic SQL statements will be recorded in the trace file.

## TraceStmtOnly CLI/ODBC configuration keyword

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceTime CLI/ODBC configuration keyword

Captures elapsed time counters in the trace file.

**db2cli.ini keyword syntax:**

TraceTime = 1 | 0

**Default setting:**

Elapsed time counters are included in the trace file.

**Only applicable when:**

the Trace option is turned on.

**Usage notes:**

When TraceTime is set to 1, elapsed time counters will be captured in the trace file.

For example:

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
  —> Time elapsed - +6.785751E+000 seconds ( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <— SQL_SUCCESS Time elapsed - +2.527400E-002 seconds
```

Turn TraceTime off, by setting it to 0, to improve performance or to generate smaller trace files. For example:

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <— SQL_SUCCESS
```

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## TraceTimestamp CLI/ODBC configuration keyword

Specifies what type of timestamp information (if any) is recorded in the CLI/ODBC trace.

**db2cli.ini keyword syntax:**

TraceTimestamp = 0 | 1 | 2 | 3

**Default setting:**

No timestamp information is written to the trace file.

**Only applicable when:**

the Trace option is turned on.

**Usage notes:**

Setting TraceTimeStamp to a value other than the default of 0 means the current timestamp or absolute execution time is added to the beginning of each line of

## TraceTimestamp CLI/ODBC configuration keyword

trace information as it is being written to the DB2 CLI trace file. The following settings indicate what type of timestamp information is captured in the trace file:

- 0 = no timestamp information
- 1 = processor ticks and ISO timestamp (absolute execution time in seconds and milliseconds, followed by a timestamp)
- 2 = processor ticks (absolute execution time in seconds and milliseconds)
- 3 = ISO timestamp

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

---

## Trusted\_Connection CLI/ODBC configuration keyword

Specifies whether a connection made with the current authenticated user is allowed.

### Syntax:

Trusted\_Connection=Yes

**Note:** This keyword will have no effect if set in the `db2cli.ini` file. It should instead be provided in the connection string to `SQLDriverConnect()`.

### Default setting:

CLI uses the user ID and password information provided in the connection string to `SQLDriverConnect()`, not the current authenticated user.

### Usage notes:

CLI applications that connect to a database will typically connect using the function `SQLDriverConnect()`. One of the input arguments for this function is the *DriverCompletion* value, which determines when a dialog will be opened. The following are the values of *DriverCompletion* :

- `SQL_DRIVER_PROMPT`: A dialog is always initiated.
- `SQL_DRIVER_COMPLETE`: A dialog is only initiated if there is insufficient information in the connection string.
- `SQL_DRIVER_COMPLETE_REQUIRED`: A dialog is only initiated if there is insufficient information in the connection string. Only mandatory information is requested. The user is prompted for required information only.
- `SQL_DRIVER_NOPROMPT`: The user is not prompted for any information. A connection is attempted with the information contained in the connection string. If there is not enough information, `SQL_ERROR` is returned.

**Note:** More details on *DriverCompletion* can be found in the documentation for `SQLDriverConnect()`.

Some applications, for example, those in a Kerberos environment, might require that a user be able to connect to a DB2 server without providing a user ID or password. If the application uses the `SQL_DRIVER_NO_PROMPT` option on the `SQLDriverConnect()` call, the connection will be attempted without the user authentication. This keyword is then not required.



## Trusted\_Connection CLI/ODBC configuration keyword

In the case where a third party application is involved and the prompt level used by the application is something other than `SQL_DRIVER_NO_PROMPT`, CLI will open a dialog to request the missing information from the user. Setting `Trusted_Connection` to `Yes`, by providing it to the input connection string for `SQLDriverConnect()` ("`Trusted_Connection=Yes`"), causes CLI to ignore any user ID or password string (including blank strings) from the connection string and ignore the prompt level of the connection function. CLI will use the current authenticated user to attempt the connection to the database. If the connection attempt fails, the user will be prompted for the user ID and password.

This keyword is used only in the connection string for `SQLDriverConnect()`; setting it in the `db2cli.ini` file will have no effect.

---

## TxnIsolation CLI/ODBC configuration keyword

Sets the default isolation level.

### db2cli.ini keyword syntax:

For Version 9.7 Fix Pack 4 or earlier fix packs:

`TxnIsolation = 1 | 2 | 4 | 8 | 32`

For Version 9.7 Fix Pack 5 or later fix packs:

`TxnIsolation = ReadUncommitted | ReadCommitted | RepeatableRead | Serializable | NoCommit`

### Default setting:

2 or `ReadCommitted` (Cursor Stability)

### Equivalent statement attribute:

`SQL_ATTR_TXN_ISOLATION`

### Usage notes:

Sets the isolation level to:

- 1 = `SQL_TXN_READ_UNCOMMITTED` - Read uncommitted (Uncommitted read)
- 2 = `SQL_TXN_READ_COMMITTED` (default) - Read committed (Cursor stability)
- 4 = `SQL_TXN_REPEATABLE_READ` - Repeatable read (Read stability)
- 8 = `SQL_TXN_SERIALIZABLE` - Serializable (Repeatable read)
- 32 = `SQL_TXN_NOCOMMIT` - (No commit, DB2 Universal Database for AS/400 only; this setting is similar to autocommit).

The words in parentheses are the IBM terminology for the equivalent SQL92 isolation levels. Note that *no commit* is not an SQL92 isolation level and is supported on IBM DB2 for IBM i only.

Table 165. Supported isolation levels

Isolation level	Keyword	SQL92	IBM terminology
1	<code>SQL_TXN_READ_UNCOMMITTED</code>	Read uncommitted	Uncommitted read
2	<code>SQL_TXN_READ_COMMITTED</code> (default)	Read committed	Cursor stability
4	<code>SQL_TXN_REPEATABLE_READ</code>	Repeatable read	Read stability

## TxnIsolation CLI/ODBC configuration keyword

Table 165. Supported isolation levels (continued)

Isolation level	Keyword	SQL92	IBM terminology
8	SQL_TXN_SERIALIZABLE	Serializable	Repeatable read
32	SQL_TXN_NOCOMMIT	Not an SQL92 isolation level	No commit

In Version 9.7 Fix Pack 5 and later Fix Packs, you can use the listed textual values to set the *TxnIsolation* keyword in the `db2cli.ini` file:

- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serializable
- NoCommit

If you use a text value that is not in the list, the value is ignored and *TxnIsolation* is set to the default value.

This keyword is only applicable if you use the default isolation level. If the application has explicitly set the isolation level for a connection or statement handle, this keyword setting is ignored.

---

## UID CLI/ODBC configuration keyword

Defines a default user ID.

**db2cli.ini keyword syntax:**

UID = *userid*

**Default setting:**

None

**Usage notes:**

The specified *userid* value is used if a *userid* is not provided by the application at connect time.

---

## Underscore CLI/ODBC configuration keyword

Specifies whether the underscore character '\_' is treated as a wildcard.

**db2cli.ini keyword syntax:**

Underscore = 0 | 1

**Default setting:**

The underscore character matches any single character or no character.

**Usage notes:**

This keyword specifies if the underscore character '\_' will be recognized as a wildcard or only as the underscore character. The possible settings are as follows:

- 0 - The underscore character is treated only as the underscore character.
- 1 - The underscore character is treated as a wildcard that matches any single character, including no character.

## Underscore CLI/ODBC configuration keyword

Setting Underscore to 0 can improve performance when there are database objects with names that contain underscores.

This keyword applies only to the following catalog functions that accept search patterns as arguments:

- SQLColumnPrivileges()
- SQLColumns()
- SQLProcedureColumns()
- SQLProcedures()
- SQLTablePrivileges()
- SQLTables()

Note that catalog functions may only accept search patterns on particular arguments. Refer to the documentation of the specific function for details.

---

## UseOldStpCall CLI/ODBC configuration keyword

Controls how cataloged procedures are invoked.

**db2cli.ini keyword syntax:**

UseOldStpCall = 0 | 1

**Default setting:**

Invokes procedures using the new CALL method where GRANT EXECUTE must be granted on the procedure.

**Usage notes:**

Prior to DB2 Universal Database Version 8, the invoker of a procedure had to have EXECUTE privilege on any package invoked from the procedure. Now, the invoker must have EXECUTE privilege on the procedure and only the definer of the procedure has to have EXECUTE privilege on any required packages.

This keyword controls which method is used to invoke the procedure. Setting UseOldStpCall on causes the procedure to be invoked using the deprecated sqlproc() API when the precompiler fails to resolve a procedure on a CALL statement. Turning this keyword off will invoke procedures where GRANT EXECUTE must be granted on the procedure.

---

## UseServerMsgSP CLI/ODBC configuration keyword

Specifies whether a stored procedure is called to retrieve message text when connected to DB2 for z/OS servers.

**db2cli.ini keyword syntax:**

UseServerMsgSP = 0 | 1

**Default setting:**

CLI does not use the server stored procedure to return messages, but uses the local message files.

**Equivalent connection attribute:**

SQL\_ATTR\_SERVER\_MSGTXT\_SP

**Usage notes:**

CLI calls the stored procedure indicated by the SQL\_ATTR\_SERVER\_MSGTXT\_SP connection attribute. If this attribute is

## UseServerMsgSP CLI/ODBC configuration keyword

not set, CLI calls the SYSIBM.SQLCAMESSAGE stored procedure. If this attribute is set to DSNACCMG, CLI calls DSNACCMG when connected to DB2 for z/OS Version 7 servers and calls SYSIBM.SQLCAMESSAGE when connected to DB2 for z/OS Version 8 or later.

Applications using this keyword should also set the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 to indicate when CLI should call this procedure to retrieve the message information from the server. If the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 is not set, then the default is to check the local message files first. See the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 for more details on the options available.

DSNACCMG has been deprecated in DB2 for z/OS Version 9 and might be removed in a future release. If SQL\_ATTR\_SERVER\_MSGTXT\_SP is set to DSNACCMG, set this attribute to a different store procedure to retrieve messages text. Alternatively, use local message files or use the ServerMsgTextSP configuration keyword.

---

## ServerMsgTextSP CLI/ODBC configuration keyword

Specifies which stored procedure is used to retrieve message text from DB2 for z/OS.

**db2cli.ini keyword syntax:**

**ServerMsgTextSP = stored procedure name**

**Default setting:**

CLI does not use the server stored procedure to return messages, but uses the local message files.

**Equivalent connection attribute:**

SQL\_ATTR\_SERVER\_MSGTXT\_SP

**Usage notes:**

Applications using this keyword should also set the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 to indicate when CLI should call this procedure to retrieve the message information from the server. If the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 is not set, then the default is to check the local message files first. See the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 for more details on the options available.

The difference between **ServerMsgTextSP** and “UseServerMsgSP CLI/ODBC configuration keyword” on page 423 is that **UseServerMsgSP** can be turned on and off to call the procedure specified in the SQL\_ATTR\_SERVER\_MSGTXT\_SP connection attribute, while **ServerMsgTextSP** needs to have the procedure explicitly specified.

---

## WarningList CLI/ODBC configuration keyword

Specifies which errors to downgrade to warnings.

**db2cli.ini keyword syntax:**

WarningList = " 'xxxx', 'yyyy', ..."

**Default setting:**

Do not downgrade any SQLSTATEs.

**Usage notes:**

Any number of SQLSTATEs returned as errors can be downgraded to warnings. Each must be delimited with single quotes, separated by commas, and in uppercase. The entire string must also be enclosed in double quotes. For example:

```
WarningList=" '01S02', 'HY090' "
```

---

## XMLDeclaration CLI/ODBC configuration keyword

Controls the generation of an XML declaration when XML data is implicitly serialized to an application variable.

### db2cli.ini keyword syntax:

```
XMLDeclaration = non-negative integer < 7 | 7
```

### Default setting:

A BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.

### Usage notes:

The XMLDeclaration keyword controls which elements of an XML declaration are prepended to an application buffer when XML data is implicitly serialized to an application buffer. This setting does not affect the result of the XMLSERIALIZE function.

The following values represent components to be generated during implicit serialization. Set this keyword by adding together the value of each component required.

- 0 No declarations or byte order marks (BOMs) are added to the output buffer.
- 1 A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16 or UTF-32. (Although a UTF-8 BOM exists, the database server does not generate it, even if the target encoding is UTF-8.)
- 2 A minimal XML declaration is generated, containing only the XML version.
- 4 An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this keyword.

For example, if you wanted a BOM and minimal XML declaration (without an encoding attribute) to be generated during implicit serialization, you would set XMLDeclaration = 3, where 3 is the sum of 1 (the value to indicate generation of a BOM) and 2 (the value to indicate generation of a minimal XML declaration).

To prevent any declarations or BOM from being generated, set XMLDeclaration as follows: XMLDeclaration = 0.



---

## Chapter 4. Environment, connection, and statement attributes in CLI applications

Environments, connections, and statements each have a defined set of attributes (or options). All attributes can be queried by the application, but only some attributes can be changed from their default values. By changing attribute values, the application can change the behavior of CLI.

An environment handle has attributes which affect the behavior of CLI functions under that environment. The application can specify the value of an attribute by calling `SQLSetEnvAttr()` and can obtain the current attribute value by calling `SQLGetEnvAttr()`. `SQLSetEnvAttr()` can only be called before any connection handles have been allocated for the environment handle. For details on each environment attribute, refer to the list of CLI environment attributes.

A connection handle has attributes which affect the behavior of CLI functions under that connection. Of the attributes that can be changed:

- Some can be set any time once the connection handle is allocated.
- Some can be set only before the actual connection has been established.
- Some can be set any time after the connection has been established.
- Some can be set after the connection has been established, but only while there are no outstanding transactions or open cursors.

The application can change the value of connection attributes by calling `SQLSetConnectAttr()` and can obtain the current value of an attribute by calling `SQLGetConnectAttr()`. An example of a connection attribute which can be set any time after a handle is allocated is the auto-commit option `SQL_ATTR_AUTOCOMMIT`. For details on each connection attribute, refer to the list of CLI connection attributes.

A statement handle has attributes which affect the behavior of CLI functions executed using that statement handle. Of the statement attributes that can be changed:

- Some attributes can be set, but currently are limited to only one specific value.
- Some attributes can be set any time after the statement handle has been allocated.
- Some attributes can only be set if there is no open cursor on that statement handle.

The application can specify the value of any statement attribute that can be set by calling `SQLSetStmtAttr()` and can obtain the current value of an attribute by calling `SQLGetStmtAttr()`. For details on each statement attribute, refer to the list of CLI statement attributes.

The `SQLSetConnectAttr()` function cannot be used to set statement attributes. This was supported in versions of CLI prior to version 5.

Many applications just use the default attribute settings; however, there may be situations where some of these defaults are not suitable for a particular user of the application. Some default values can be changed by setting the CLI/ODBC configuration keywords. CLI provides end users with two methods of setting some

## Environment, connection, and statement attributes in CLI applications

configuration keywords. The first method is to specify the keyword and its new default attribute value(s) in the connection string input to the `SQLDriverConnect()` and `SQLBrowseConnect()` functions. The second method involves the specification of the new default attribute value(s) in a CLI initialization file using CLI/ODBC configuration keywords.

The CLI initialization file can be used to change default values for all CLI applications on that workstation. This may be the end user's only means of changing the defaults if the application does not provide a means for the user to provide default attribute values in the `SQLDriverConnect()` connection string. Default attribute values that are specified on `SQLDriverConnect()` override the values in the CLI initialization file for that particular connection.

The mechanisms for changing defaults are intended for end user tuning; application developers must use the appropriate set-attribute function. If an application does call a set-attribute or option function with a value different from the initialization file or the connection string specification, then the initial default value is overridden and the new value takes effect.

The diagram below shows the addition of the attribute functions to the basic connect scenario.



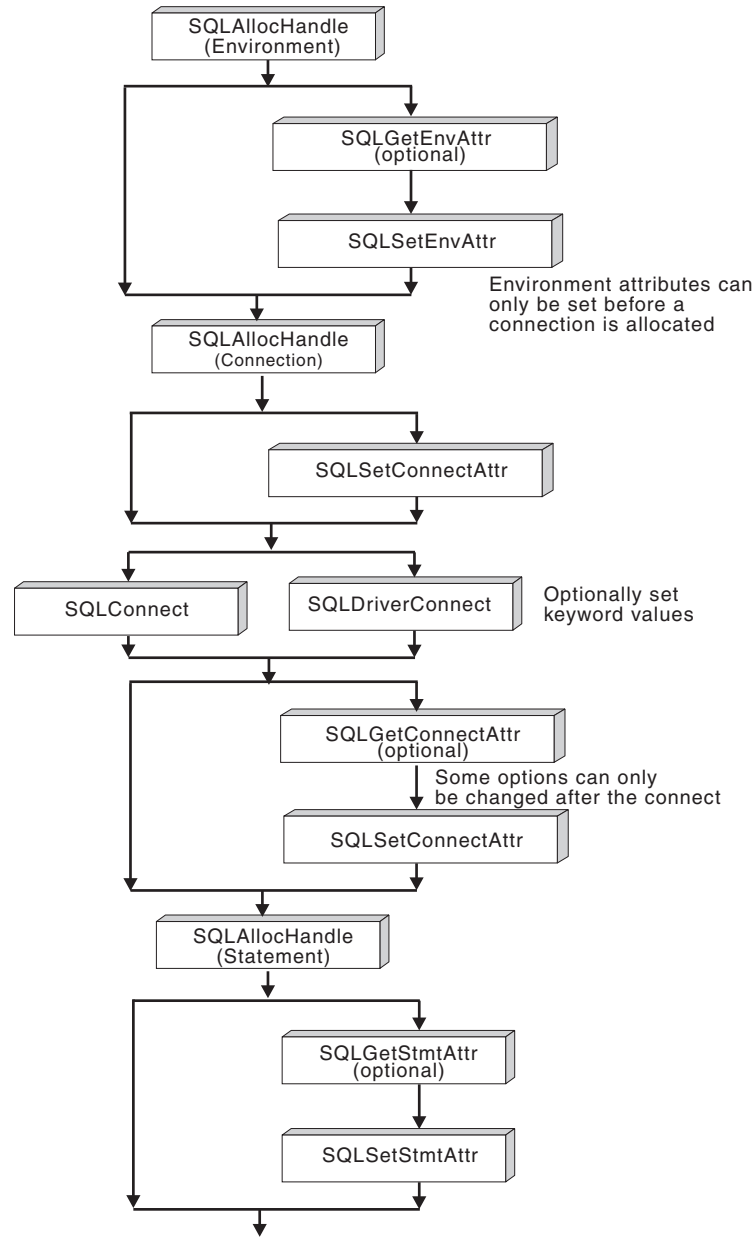


Figure 1. Setting and retrieving attributes (options)

## Environment attributes (CLI) list

CLI environment attributes that can be set using the `SQLSetEnvAttr()`.

ODBC does not support setting driver-specific environment attributes using `SQLSetEnvAttr()`. Only CLI applications can set the CLI-specific environment attributes using this function.

### SQL\_ATTR\_CONNECTION\_POOLING

This attribute was deprecated in DB2 UDB for Linux, UNIX, and Windows Version 8.

This attribute is not supported when accessing the Informix database server.

## Environment attributes (CLI) list

### SQL\_ATTR\_CONNECTTYPE

This attribute replaces the SQL\_CONNECTTYPE attribute. A 32-bit integer value that specifies whether this application is to operate in a coordinated or uncoordinated distributed environment. The possible values are:

- **SQL\_CONCURRENT\_TRANS:** The application can have concurrent multiple connections to any one database or to multiple databases. Each connection has its own commit scope. No effort is made to enforce the coordination of the transaction. If an application issues a commit by using the environment handle on `SQLEndTran()` and not all of the connections commit successfully, the application is responsible for recovery. This is the default.
- **SQL\_COORDINATED\_TRANS:** The application can coordinate commit and rollbacks among multiple database connections. This option setting corresponds to the specification of the Type 2 CONNECT in embedded SQL. In contrast to the `SQL_CONCURRENT_TRANS` setting that was previously described, the application is permitted only one open connection per database.

**Note:** This connection type results in the default for the `SQL_ATTR_AUTOCOMMIT` connection option to be `SQL_AUTOCOMMIT_OFF`.

If you change this attribute from the default, you must set it before any connections are established on the environment handle.

Application typically set this attribute as an environment attribute with a call to `SQLSetEnvAttr()` function. The `SQLSetEnvAttr()` function is called as soon as the environment handle is allocated. However, because ODBC applications cannot access `SQLSetEnvAttr()` function, ODBC applications must set this attribute using `SQLSetConnectAttr()` function after each connection handle is allocated, but before any connections are established.

All connections on an environment handle must have the same `SQL_ATTR_CONNECTTYPE` setting. An environment cannot have both concurrent and coordinated connections. The type of the first connection determines the type of all subsequent connections. `SQLSetEnvAttr()` returns an error if an application attempts to change the connection type while there is an active connection.

You can also set the default connect type by using the “ConnectType CLI/ODBC configuration keyword” on page 343.

The `SQL_ATTR_CONNECTTYPE` attribute is an IBM defined extension.

### SQL\_ATTR\_CP\_MATCH

This attribute was deprecated in DB2 database version 8.

This attribute is not supported when accessing the Informix database server.

### SQL\_ATTR\_DIAGLEVEL

#### Description

A 32-bit integer value which represents the diagnostic level. This is equivalent to the database manager `DIAGLEVEL` parameter.

#### Values

Valid values are: 0, 1, 2, 3, or 4. (The default value is 3.)

### Usage notes

You must set this attribute before any connection handles are created.

### SQL\_ATTR\_DIAGPATH

#### Description

A pointer to a null-terminated character string that contains the name of the directory where diagnostic data is to be placed. This is equivalent to the database manager DIAGPATH parameter.

#### Values

The default value is the db2dump directory on UNIX and Linux operating systems, and the db2 directory on Windows operating systems.

### Usage notes

You must set this attribute before any connection handles are created.

### SQL\_ATTR\_INFO\_ACCTSTR

#### Description

A pointer to a null-terminated character string that is used to identify the client accounting string that is sent to the data server when DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

#### Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 200 characters. In DB2 Version 9.7 Fix Pack 6 and later, a CLI application can set the SQL\_ATTR\_INFO\_ACCTSTR attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

The SQL\_ATTR\_INFO\_ACCTSTR attribute is an IBM-defined extension.

### SQL\_ATTR\_INFO\_APPLNAME

#### Description

A pointer to a null-terminated character string that is used to identify the client application name that is sent to the data server when DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

#### Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 32 characters. In DB2 Version 9.7 Fix Pack 6 and later, a CLI application can set the SQL\_ATTR\_INFO\_APPLNAME attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

The SQL\_ATTR\_INFO\_APPLNAME attribute is an IBM defined extension.

## Environment attributes (CLI) list

### SQL\_ATTR\_INFO\_USERID

#### Description

A pointer to a null-terminated character string that is used to identify the client user ID that is sent to the data server when DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

#### Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 16 characters. In DB2 Version 9.7 Fix Pack 6 and later, a CLI application can set the SQL\_ATTR\_INFO\_USERID attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.

Do not confuse the client user ID with the authentication user ID. The client user ID is for identification purposes only and is not used for any authorization. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

The SQL\_ATTR\_INFO\_USERID attribute is an IBM defined extension.

### SQL\_ATTR\_INFO\_WRKSTNNAME

#### Description

A pointer to a null-terminated character string that is used to identify the client workstation name that is sent to the data server when using DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

#### Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 18 characters. In DB2 Version 9.7 Fix Pack 6 and later, a CLI application can set the SQL\_ATTR\_INFO\_WRKSTNNAME attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (\_) or period (.).

The SQL\_ATTR\_INFO\_WRKSTNNAME attribute is an IBM defined extension.

In DB2 Version 9.7 Fix pack 6 and later, if **SQL\_ATTR\_INFO\_WRKSTNNAME** attribute is not specified, default value that consists of the host name is used. The host name is obtained by gethostname() function call. If host name is not configured or an error is encountered during the gethostname() function call, no value for the **SQL\_ATTR\_INFO\_WRKSTNNAME** attribute is sent to the server.

### SQL\_ATTR\_MAXCONN

This attribute was deprecated in DB2 Version 8.

This attribute is not supported when accessing the Informix database servers.

### SQL\_ATTR\_NOTIFYLEVEL

**Description**

A 32-bit integer value that represents the notification level. This is equivalent to the database manager NOTIFYLEVEL parameter.

**Values**

Valid values are: 0, 1, 2, 3, or 4. (The default value is 3.)

**Usage notes**

You must set this attribute value before any connection handles are created.

This attribute is not supported when accessing the Informix database servers.

**SQL\_ATTR\_ODBC\_VERSION**

**Description**

A 32-bit integer that determines whether certain functionality exhibits ODBC 2.x (CLI v2) behavior or ODBC 3.0 (CLI v5) behavior. ODBC applications must set this environment attribute before calling any function that has an SQLHENV argument, or the call will return SQLSTATE HY010 (Function sequence error.).

**Values**

The listed values are used to set the value of this attribute:

- SQL\_OV\_ODBC3: Causes the listed ODBC 3.0 (CLI v5) behavior:
  - CLI returns and expects ODBC 3.0 (CLI v5) codes for date, time, and timestamp.
  - CLI returns ODBC 3.0 (CLI v5) SQLSTATE codes when SQLError(), SQLGetDiagField(), or SQLGetDiagRec() functions are called.
  - The *CatalogName* argument in a call to SQLTables() function accepts a search pattern.
- SQL\_OV\_ODBC2: Causes the listed ODBC 2.x (CLI v2) behavior:
  - CLI returns and expects ODBC 2.x (CLI v2) codes for date, time, and timestamp.
  - CLI returns ODBC 2.0 (CLI v2) SQLSTATE codes when SQLError(), SQLGetDiagField(), or SQLGetDiagRec() functions are called.
  - The *CatalogName* argument in a call to SQLTables() function does not accept a search pattern.
- SQL\_OV\_ODBC3\_80: Causes the listed ODBC 3.0 (CLI v5) behavior:
  - CLI returns and expects ODBC 3.x codes for date, time, and timestamp.
  - CLI returns ODBC 3.x SQLSTATE codes when SQLError(), SQLGetDiagField(), or SQLGetDiagRec() functions are called.
  - The *CatalogName* argument in a call to SQLTables() function accepts a search pattern.

**SQL\_ATTR\_OUTPUT\_NTS**

**Description**

A 32-bit integer value that controls the use of null-termination in output arguments.

**Values**

The possible values are:

## Environment attributes (CLI) list

- **SQL\_TRUE**: CLI uses null termination to indicate the length of output character strings (default).
- **SQL\_FALSE**: CLI does not use null termination in output character strings.

The CLI functions that are affected by this attribute are all of the functions that are called for the environment (and for any connections and statements that are allocated under the environment) that have character string parameters.

You can only set this attribute when there are no connection handles that are allocated under this environment.

### SQL\_ATTR\_PROCESSCTL

#### Description

A 32-bit mask that sets process-level attributes, which affect all environments and connections for the process. You must set this attribute before the environment handle is allocated.

The call to `SQLSetEnvAttr()` must have the *EnvironmentHandle* argument set to `SQL_NULL_HANDLE`. The settings remain in effect for the duration of the process. Generally, use this attribute only for performance sensitive applications, where large numbers of CLI function calls are being made. Before setting any of these bits, ensure that the application, and any other libraries that the application calls, comply with the restrictions that are listed.

#### Values

You can combine the listed values to form a bit mask:

- **SQL\_PROCESSCTL\_NOTHREAD** - This bit indicates that the application does not use multiple threads, or if it does use multiple threads, guarantees that all DB2 calls are serialized by the application. If set, CLI does not make any system calls to serialize calls to CLI, and sets the DB2 context type to `SQL_CTX_ORIGINAL`.
- **SQL\_PROCESSCTL\_NOFORK** - This bit indicates that the application will never fork a child process. By default, CLI does not check to see if an application forks a child process. However, if the `CheckForFork` CLI/ODBC configuration keyword is set, CLI checks the current process ID for each function call for all applications that are connecting to the database for which the keyword is enabled. You can set this attribute so that CLI does not check for forked processes for that application.

The `SQL_ATTR_PROCESSCTL` attribute is an IBM defined extension.

### SQL\_ATTR\_RESET\_CONNECTION

#### Description

A 32-bit unsigned integer value that specifies whether the ODBC Driver Manager notifies the ODBC drivers that a connection has been placed in the connection pool on Windows operating systems. If the `SQL_ATTR_ODBC_VERSION` environment attribute is set to `SQL_OV_ODBC3_80`, the ODBC Driver Manager sets this attribute before placing a connection in the connection pool so that the

driver can reset the other connection attributes to their default values. This connection attribute is available in Version 9.7 Fix Pack 5 and later fix packs.

### Values

The only possible value is:

- **SQL\_RESET\_CONNECTION\_YES** (default): The ODBC Driver Manager notifies the ODBC drivers that a connection has been placed in the connection pool.

**Note:** You should use **SQL\_ATTR\_RESET\_CONNECTION** only for communication between the ODBC Driver Manager and an ODBC driver. You should not set this attribute from an application because all connection attributes will be reset to their default value. For example, any connection attribute values that you set by using the **SQLSetConnectAttr ()** function will be reset to CLI default values and your application could behave unexpectedly.

### **SQL\_ATTR\_SYNC\_POINT**

This attribute was deprecated in DB2 database version 8.

This attribute is not supported when accessing the Informix database servers.

### **SQL\_ATTR\_TRACE**

#### Description

A pointer to a null-terminated character string that is used to turn on the CLI/ODBC trace facility.

#### Values

The string must include the CLI keywords **TRACE** and **TRACEPATHNAME**. For example:

```
"TRACE=1; TRACEPATHNAME=<dir>";
```

#### Usage notes

This attribute is not supported when accessing the Informix database servers.

### **SQL\_ATTR\_TRACENOHEADER**

#### Description

A 32-bit integer value that specifies whether header information is included in the CLI trace file.

#### Values

The possible values are:

- **0** - Header information is included in the CLI trace file.
- **1** - No header information is included in the CLI trace file.

You can use the **SQL\_ATTR\_TRACENOHEADER** attribute with an **SQL\_NULL\_HANDLE** or with a valid environment handle.

### **SQL\_ATTR\_USE\_2BYTES\_OCTET\_LENGTH**

This attribute is deprecated in DB2 database version 8.

This attribute is not supported when accessing the Informix database servers.

### **SQL\_ATTR\_USE\_LIGHT\_OUTPUT\_SQLDA**

Setting this attribute is equivalent to setting the connection attribute **SQL\_ATTR\_DESCRIBE\_OUTPUT\_LEVEL** to 0.

## Environment attributes (CLI) list

SQL\_ATTR\_USE\_LIGHT\_OUTPUT\_SQLDA is deprecated and applications should now use the connection attribute SQL\_ATTR\_DESCRIBE\_OUTPUT\_LEVEL.

### SQL\_ATTR\_USER\_REGISTRY\_NAME

#### Description

This attribute is used only when authenticating a user on a server that is using an identity mapping service.

#### Values

The SQL\_ATTR\_USER\_REGISTRY\_NAME attribute is set to a user defined string that names an identity mapping registry. The format of the name varies depending on the identity mapping service. By providing this attribute you tell the server that the user name that is provided can be found in this registry.

After setting this attribute, the value is used on subsequent attempts to establish a normal connection, establish a trusted connection, or switch the user ID on a trusted connection.

#### Usage notes

This attribute is not supported when accessing the Informix database servers.

### SQL\_CONNECTTYPE

This *Attribute* is replaced with SQL\_ATTR\_CONNECTTYPE.

### SQL\_MAXCONN

This *Attribute* is replaced with SQL\_ATTR\_MAXCONN.

### SQL\_SYNC\_POINT

This *Attribute* is replaced with SQL\_ATTR\_SYNC\_POINT.

This attribute is not supported when accessing the Informix database servers.

---

## Connection attributes (CLI) list

The following table indicates when each of the CLI connection attributes can be set. A "Yes" in the "After statements allocated" column means that the connection attribute can be set both before and after the statements are allocated.

Table 166. When connection attributes can be set

Attribute	Before connection	After connection	After statements allocated
SQL_ATTR_ACCESS_MODE	Yes	Yes	Yes <sup>a</sup>
SQL_ATTR_ALLOW_INTERLEAVED_GETDATA	Yes	Yes	Yes
SQL_ATTR_ANSI_APP	Yes	No	No
SQL_ATTR_APP_USES_LOB_LOCATOR	Yes	Yes	Yes <sup>c</sup>
SQL_ATTR_APPEND_FOR_FETCH_ONLY	Yes	Yes	No
SQL_ATTR_ASYNC_ENABLE	Yes	Yes	Yes <sup>a</sup>
SQL_ATTR_AUTO_IPD (read-only)	No	No	No
SQL_ATTR_AUTOCOMMIT	Yes	Yes	Yes <sup>b</sup>
SQL_ATTR_CLIENT_CODEPAGE	Yes	No	No
SQL_ATTR_COLUMNWISE_MRI	Yes	Yes	Yes <sup>a</sup>
SQL_ATTR_COMMITONEOF	Yes	Yes	No
SQL_ATTR_CONCURRENT_ACCESS_RESOLUTION	Yes	Yes	Yes <sup>a</sup>



Table 166. When connection attributes can be set (continued)

Attribute	Before connection	After connection	After statements allocated
SQL_ATTR_CONN_CONTEXT	Yes	No	No
SQL_ATTR_CONNECT_NODE	Yes	No	No
SQL_ATTR_CONNECTION_DEAD (read-only)	No	No	No
SQL_ATTR_CONNECTTYPE	Yes	No	No
SQL_ATTR_CURRENT_CATALOG (read-only)	No	No	No
SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION	Yes	Yes	Yes
SQL_ATTR_CURRENT_PACKAGE_PATH	Yes	Yes	Yes
SQL_ATTR_CURRENT_PACKAGE_SET	Yes	Yes <sup>a</sup>	No <sup>*</sup>
SQL_ATTR_CURRENT_SCHEMA	Yes	Yes	Yes
SQL_ATTR_DB2_APPLICATION_HANDLE (read-only)	No	No	No
SQL_ATTR_DB2_APPLICATION_ID (read-only)	No	No	No
SQL_ATTR_DB2_SQLERRP (read-only)	No	No	No
SQL_ATTR_DB2EXPLAIN	No	Yes	Yes
SQL_ATTR_DECFLOAT_ROUNDING_MODE	Yes	Yes	Yes
SQL_ATTR_DESCRIBE_CALL	Yes	Yes	Yes <sup>a</sup>
SQL_ATTR_DESCRIBE_OUTPUT_LEVEL	Yes	Yes	No
SQL_ATTR_ENLIST_IN_DTC	No	Yes	Yes
SQL_ATTR_EXTENDED_INDICATORS	No	Yes	Yes
SQL_ATTR_FET_BUF_SIZE	Yes	No	No
SQL_ATTR_FREE_LOCATORS_ON_FETCH	Yes	Yes	Yes
SQL_ATTR_FORCE_ROLLBACK	Yes	Yes	Yes
SQL_ATTR_GET_LATEST_MEMBER	No	Yes	Yes
SQL_ATTR_INFO_ACCTSTR	Yes <sup>d</sup>	Yes	Yes
SQL_ATTR_INFO_APPLNAME	Yes <sup>d</sup>	Yes	Yes
SQL_ATTR_INFO_PROGRAMID	Yes <sup>d</sup>	Yes	Yes <sup>a</sup>
SQL_ATTR_INFO_PROGRAMNAME	Yes	No	No
SQL_ATTR_INFO_USERID	Yes <sup>d</sup>	Yes	Yes
SQL_ATTR_INFO_WRKSTNNAME	Yes <sup>d</sup>	Yes	Yes
SQL_ATTR_KEEP_DYNAMIC	No	Yes	Yes
SQL_ATTR_LOB_CACHE_SIZE	Yes	Yes	Yes <sup>c</sup>
SQL_ATTR_LOGIN_TIMEOUT	Yes	No	No
SQL_ATTR_LONGDATA_COMPAT	Yes	Yes	Yes
SQL_ATTR_MAX_LOB_BLOCK_SIZE	Yes	Yes	Yes <sup>c</sup>
SQL_ATTR_MAPCHAR	Yes	Yes	Yes
SQL_ATTR_NETWORK_STATISTICS	Yes	Yes	Yes
SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE	No	Yes	No
SQL_ATTR_OVERRIDE_CODEPAGE	No	Yes	No
SQL_ATTR_PARC_BATCH	Yes	Yes	Yes <sup>*</sup>
SQL_ATTR_PING_DB (read only)	No	No	No
SQL_ATTR_PING_NTIMES	Yes	Yes	Yes
SQL_ATTR_PING_REQUEST_PACKET_SIZE	Yes	Yes	Yes
SQL_ATTR_QUIET_MODE	Yes	Yes	Yes
SQL_ATTR_RECEIVE_TIMEOUT	Yes	Yes	Yes
SQL_ATTR_REOPT	No	Yes	Yes <sup>c</sup>
SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB	Yes	Yes	Yes
SQL_ATTR_REPORT_SEAMLESSFAILOVER_WARNING	Yes	Yes	Yes <sup>*</sup>
SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN	Yes	Yes	Yes
SQL_ATTR_RETRYONERROR	Yes	Yes	Yes
SQL_ATTR_SERVER_MSGTXT_MASK	Yes	Yes	Yes
SQL_ATTR_SERVER_MSGTXT_SP	Yes	Yes	Yes
SQL_ATTR_SESSION_TIME_ZONE	Yes	No	No

## Connection attributes (CLI) list

Table 166. When connection attributes can be set (continued)

Attribute	Before connection	After connection	After statements allocated
SQL_ATTR_SQLCODEMAP	Yes	No	No
SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB	Yes	Yes	Yes
SQL_ATTR_STMT_CONCENTRATOR	Yes	Yes	Yes
SQL_ATTR_STREAM_GETDATA	Yes	Yes	Yes <sup>c</sup>
SQL_ATTR_TRUSTED_CONTEXT_PASSWORD	No	Yes	Yes
SQL_ATTR_TRUSTED_CONTEXT_USERID	No	Yes	Yes
SQL_ATTR_TXN_ISOLATION	No	Yes <sup>b</sup>	Yes <sup>a</sup>
SQL_ATTR_USE_TRUSTED_CONTEXT	Yes	No	No
SQL_ATTR_USER_REGISTRY_NAME	Yes	No	No
SQL_ATTR_WCHARTYPE	Yes	Yes <sup>b</sup>	Yes <sup>b</sup>
SQL_ATTR_XML_DECLARATION	Yes	Yes	Yes <sup>a</sup>

<sup>a</sup> Only affect subsequently allocated statements.

<sup>b</sup> Attribute can be set only if there are no open transactions on the connection.

<sup>c</sup> Attribute can be set only if there are no open cursors on the connection. The attribute affects all statements.

<sup>d</sup> In DB2 Version 9.7 Fix Pack 6 and later client, you can set the attribute on DB2 for i V6R1 and later. The attribute can also be set on DB2 for z/OS, DB2 for Linux, UNIX, and Windows, and IBM Informix from the DB2 Version 9.7 GA client.

\* Setting this attribute after statements have been allocated will not result in an error. However, determining which packages are used by which statements are ambiguous and unexpected behavior might occur. You should not set this attribute after statements have been allocated.

### Attribute

#### ValuePtr contents

#### SQL\_ATTR\_ACCESS\_MODE

A 32-bit integer value which can be either:

- **SQL\_MODE\_READ\_ONLY**: the application is indicating that it will not be performing any updates on data from this point on. Therefore, a less restrictive isolation level and locking can be used on transactions: uncommitted read (SQL\_TXN\_READ\_UNCOMMITTED). CLI does not ensure that requests to the database are *read-only*. If an update request is issued, CLI will process it using the transaction isolation level it has selected as a result of the SQL\_MODE\_READ\_ONLY setting.
- **SQL\_MODE\_READ\_WRITE (default)**: the application is indicating that it will be making updates on data from this point on. CLI will go back to using the default transaction isolation level for this connection.

There must not be any outstanding transactions on this connection.

#### SQL\_ATTR\_ALLOW\_INTERLEAVED\_GETDATA

Specifies whether the application can call SQLGetData() for previously accessed LOB columns and maintain the data offset position from the previous call to SQLGetData() when querying data servers that support Dynamic Data Format. This attribute has one of the following values:

- **SQL\_ALLOW\_INTERLEAVED\_GETDATA\_OFF** - This default setting does not allow applications to call SQLGetData() for previously accessed LOB columns.
- **SQL\_ALLOW\_INTERLEAVED\_GETDATA\_ON** - This keyword only affects connections to database servers that support Dynamic Data Format, also known as progressive streaming. Specify this option to

allow applications to call `SQLGetData()` for previously accessed LOB columns and start reading LOB data from where the application stopped reading during the previous read.

Setting the “AllowInterleavedGetData CLI/ODBC configuration keyword” on page 325 is an alternative method of specifying this behavior at the connection level.

### SQL\_ATTR\_ANSI\_APP

A 32-bit unsigned integer that identifies an application as an ANSI or Unicode application. This attribute has either of the following values:

- **SQL\_AA\_TRUE (default):** the application is an ANSI application. All character data is passed to and from the application in the native application (client) codepage using the ANSI version of the CLI/ODBC functions.
- **SQL\_AA\_FALSE:** the application is a Unicode application. All character data is passed to and from the application in Unicode when the Unicode (W) versions of the CLI/ODBC functions are called.

### SQL\_ATTR\_APP\_USES\_LOB\_LOCATOR

A 32-bit unsigned integer that indicates if applications are using LOB locators. This attribute has either of the following values:

- **1 (default):** Indicates that applications are using LOB locators.
- **0:** For applications that do not use LOB locators and are querying data on a server that supports Dynamic Data Format, also known as progressive streaming, specify 0 to indicate that LOB locators are not used and allow the return of LOB data to be optimized.

This keyword is ignored for stored procedure result sets.

If the keyword is set to 0 and an application binds a LOB locator to a result set using `SQLBindCol()`, an Invalid conversion error will be returned by the `SQLFetch()` function.

Setting the “AppUsesLOBLocator CLI/ODBC configuration keyword” on page 327 is an alternative method of specifying this behavior.

### SQL\_ATTR\_APPEND\_FOR\_FETCH\_ONLY

By default, CLI appends the “FOR FETCH ONLY” clause to read SELECT statements when connected to DB2 for z/OS or IBM DB2 for IBM i (DB2 for i) databases.

This attribute allows an application to control at a connection level when CLI appends the “FOR FETCH ONLY” clause. For example, an application is binding the CLI packages using different bind BLOCKING options (for example, BLOCKING UNAMBIG) and wants to suppress the blocking in order to keep positioned on a given row.

To change the default CLI behavior, the keyword is set as follows:

- **0:** CLI never appends the “FOR FETCH ONLY” clause to read SELECT statements regardless of the server type it is connecting to.
- **1:** CLI always appends the “FOR FETCH ONLY” clause to read SELECT statements regardless of the server type it is connecting to.

The attribute should be set either after the connection is allocated or immediately after it is established and should be set once for the duration of the execution of the application. Application can query the attribute with `SQLGetConnectAttr()` after connection is established or after this attribute is set.

## Connection attributes (CLI) list

Setting the “AppendForFetchOnly CLI/ODBC configuration keyword” on page 328 is an alternative method of specifying this behavior.

### SQL\_ATTR\_ASYNC\_ENABLE

A 32-bit integer value that specifies whether a function called with a statement on the specified connection is executed asynchronously:

- **SQL\_ASYNC\_ENABLE\_OFF (default)** = Off
- **SQL\_ASYNC\_ENABLE\_ON** = On

Setting **SQL\_ASYNC\_ENABLE\_ON** enables asynchronous execution for all statement handles allocated on this connection. An error is returned if asynchronous execution is turned on while there is an active statement on the connection.

This attribute can be set whether `SQLGetInfo()`, called with the *InfoType* `SQL_ASYNC_MODE`, returns `SQL_AM_CONNECTION` or `SQL_AM_STATEMENT`.

Once a function has been called asynchronously, only the original function, `SQLAllocHandle()`, `SQLCancel()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` can be called on the statement or the connection associated with *StatementHandle*, until the original function returns a code other than `SQL_STILL_EXECUTING`. Any other function called on *StatementHandle* or the connection associated with *StatementHandle* returns `SQL_ERROR` with an `SQLSTATE` of `HY010` (Function sequence error).

The following functions can be executed asynchronously:

`SQLBulkOperations()`, `SQLColAttribute()`, `SQLColumnPrivileges()`, `SQLColumns()`, `SQLDescribeCol()`, `SQLDescribeParam()`, `SQLExecDirect()`, `SQLExecute()`, `SQLExtendedFetch()`, `SQLExtendedPrepare()`, `SQLFetch()`, `SQLFetchScroll()`, `SQLForeignKeys()`, `SQLGetData()`, `SQLGetLength()`, `SQLGetPosition()`, `SQLMoreResults()`, `SQLNumResultCols()`, `SQLParamData()`, `SQLPrepare()`, `SQLPrimaryKeys()`, `SQLProcedureColumns()`, `SQLProcedures()`, `SQLRowCount()`, `SQLSetPos()`, `SQLSpecialColumns()`, `SQLStatistics()`, `SQLTablePrivileges()`, `SQLTables()`.

**Note:** Any Unicode equivalent of a function stated above can be called asynchronously.

### SQL\_ATTR\_AUTO\_IPD

A read-only 32-bit unsigned integer value that specifies whether automatic population of the IPD after a call to `SQLPrepare()` is supported:

- **SQL\_TRUE** = Automatic population of the IPD after a call to `SQLPrepare()` is supported by the server.
- **SQL\_FALSE** = Automatic population of the IPD after a call to `SQLPrepare()` is not supported by the server. Servers that do not support prepared statements will not be able to populate the IPD automatically.

If **SQL\_TRUE** is returned for the `SQL_ATTR_AUTO_IPD` connection attribute, the statement attribute `SQL_ATTR_ENABLE_AUTO_IPD` can be set to turn automatic population of the IPD on or off. If `SQL_ATTR_AUTO_IPD` is **SQL\_FALSE**, `SQL_ATTR_ENABLE_AUTO_IPD` cannot be set to **SQL\_TRUE**.

The default value of `SQL_ATTR_ENABLE_AUTO_IPD` is equal to the value of `SQL_ATTR_AUTO_IPD`.

This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`.

### SQL\_ATTR\_AUTOCOMMIT

A 32-bit unsigned integer value that specifies whether to use auto-commit or manual commit mode:

- **SQL\_AUTOCOMMIT\_OFF**: the application must manually, explicitly commit or rollback transactions with `SQLEndTran()` calls.
- **SQL\_AUTOCOMMIT\_ON (default)**: CLI operates in auto-commit mode by default. Each statement is implicitly committed. Each statement that is not a query is committed immediately after it has been executed or rolled back if failure occurred. Each query is committed immediately after the associated cursor is closed.

**Note:** If this is a coordinated distributed unit of work connection, then the default is **SQL\_AUTOCOMMIT\_OFF**

Since in many DB2 environments, the execution of the SQL statements and the commit might be flowed separately to the database server, autocommit can be expensive. It is recommended that the application developer take this into consideration when selecting the auto-commit mode.

**Note:** Changing from manual commit to auto-commit mode will commit any open transaction on the connection.

### SQL\_ATTR\_CLIENT\_CODEPAGE

A 32-bit unsigned integer value that specifies connection level codepage. Specifying this attribute will override any environment level default codepage setting.

*Example 1:* Setting the codepage to be used by this database connection

```
SQLINTEGER iUnicode = 1208;
cliRC = SQLSetConnectAttr(hdbc,
    SQL_ATTR_CLIENT_CODEPAGE,
    (SQLPOINTER) iUnicode,
    SQL_IS_INTEGER);
```

### SQL\_ATTR\_CLIENT\_LOB\_BUFFERING

Specifies whether LOB locators or the underlying LOB data is returned in a result set for LOB columns that are not bound. By default, locators are returned. If an application usually fetches unbound LOBs and then must retrieve the underlying LOB data, the application's performance can be improved by retrieving the LOB data from the outset; this reduces the number of synchronous waits and network flows. The possible values for this attribute are:

- **SQL\_CLIENTLOB\_USE\_LOCATORS** (default) - LOB locators are returned
- **SQL\_CLIENTLOB\_BUFFER\_UNBOUND\_LOBS** - actual LOB data is returned

### SQL\_ATTR\_CLIENT\_TIME\_ZONE

A null-terminated character string in the format  $\pm hh:mm$ , containing the Time Zone information. Specifying this attribute will override the default Operating System Time Zone value of Client host.

### SQL\_ATTR\_COLUMNWISE\_MRI

A 32-bit unsigned integer that enables CLI applications connected to DB2 for z/OS servers to convert array input chaining into column-wise array input for INSERT operations. This attribute is available starting in Version 9.7 Fix Pack 5. The possible values are as follows:

## Connection attributes (CLI) list

- **SQL\_COLUMNWISE\_MRI\_OFF (default):** CLI does not convert chaining data to column-wise array input.
- **SQL\_COLUMNWISE\_MRI\_ON:** CLI converts array input chaining to column-wise array input. The Multi-Row Insert (MRI) feature in DB2 for z/OS expects data to be in column-wise array form. If your application uses array input chaining, this conversion helps you optimize your application performance because data is sent in a compact array form each time you call `SQLExecute()`. For more information about array input chaining, see `SQL_ATTR_CHAINING_BEGIN`.

For non-DB2 for z/OS servers, CLI automatically converts chaining data to row-wise array input and setting this attribute has no effect.

The conversion is not performed in the following cases:

- Bind parameters with a LOB data type such as `SQL_CLOB`, `SQL_BLOB`, `SQL_LONGVARIABLE`, `SQL_LONGVARGRAPHIC`, `SQL_DBCLOB`, or `SQL_XML`.
- Bind input data-at-execute parameters by setting their value to `SQL_DATA_AT_EXEC` to pass data to INSERT operations by calling the `SQLPutData()` and `SQLParamData()` functions.
- Space to store all the application data in the internal buffers is not available.

### **SQL\_ATTR\_COMMITONEOF**

A 32-bit integer value that specifies whether or not to issue an implicit COMMIT immediately after reading the an entire result set and receiving an EOF. This connection attribute is available starting in Version 9.7 Fix Pack 5. The possible values for this attribute are:

- **SQL\_COMMITONEOF\_OFF (default):** A COMMIT is not implicitly issued after reading the entire result set. You have to explicitly call the `SQLFreeStmt()` function to close the cursor and release resources.
- **SQL\_COMMITONEOF\_ON:** An implicit COMMIT is issued after reading the entire result set.

**Note:** Usage of this attribute does not replace the required call to the `SQLFreeStmt()` function.

### **SQL\_ATTR\_CONCURRENT\_ACCESS\_RESOLUTION**

A 32-bit integer value that specifies the concurrent access resolution to use at the statement level. This setting overrides the default behavior specified for cursor stability (CS) scans.

- 0 = No setting. The client does not supply a prepare option.
- 1 = Use currently committed semantics. CLI flows "currently committed" on every prepare, which means that the database manager can use the currently committed version of the data for applicable scans when the data is in the process of being updated or deleted. Rows in the process of being inserted can be skipped. This setting applies when the isolation level in effect is Cursor Stability or Read Stability (for Read Stability it skips uncommitted inserts only) and is ignored otherwise. Applicable scans include read-only scans that can be part of a read-only statement as well as a non read-only statement. The settings for the registry variables `DB2_EVALUNCOMMITTED`, `DB2_SKIPDELETED`, and `DB2_SKIPINSERTED` do not apply to scans using currently committed. However, the settings for these registry variables still apply to scans that do not use currently committed.

- 2 = Wait for outcome. CLI flows "wait for outcome" on every prepare, which means that Cursor Stability and higher scans wait for the commit or rollback when encountering data in the process of being updated or deleted. Rows in the process of being inserted are not skipped. The settings for the registry variables DB2\_EVALUNCOMMITTED, DB2\_SKIPDELETED, and DB2\_SKIPINSERTED no longer apply.
- 3 = Skip locked data. CLI flows "skip locked data" on every prepare, which means that currently committed semantics are used and rows in the process of being inserted are skipped. This option is not supported on DB2 Database for Linux, UNIX, and Windows. If specified, this setting is ignored.

For DB2 Database for Linux, UNIX, and Windows, use this attribute to override the default behavior for currently committed that is defined by the **cur\_commit** configuration parameter. For DB2 for z/OS, use this attribute to enable currently committed behavior. There is no equivalent database configuration parameter available on DB2 for z/OS for specifying this behavior.

Setting the "ConcurrentAccessResolution CLI/ODBC configuration keyword" on page 340 is an alternative method of specifying this behavior.

### SQL\_ATTR\_CONN\_CONTEXT

Indicates which context the connection should use. An SQLPOINTER to either:

- a valid context (allocated by the `sqlBeginCtx()` DB2 API) to set the context
- a NULL pointer to reset the context

This attribute can only be used when the application is using the DB2 context APIs to manage multi-threaded applications. By default, CLI manages contexts by allocating one context per connection handle, and ensuring that any executing thread is attached to the correct context.

For more information about contexts, refer to the `sqlBeginCtx()` API.

This attribute is not supported when accessing Informix database server.

### SQL\_ATTR\_CONNECT\_NODE

A 32-bit integer that specifies the target logical partition of a DB2 Enterprise Server Edition database partition server that you want to connect to. The possible values for this attribute are:

- an integer between 0 and 999
- `SQL_CONN_CATALOG_NODE`

If this variable is not set, the target logical node defaults to the logical node which is defined with port 0 on the machine.

This attribute is not supported when accessing Informix database server.

There is also a corresponding keyword, the "ConnectNode CLI/ODBC configuration keyword" on page 341.

### SQL\_ATTR\_CONNECTION\_DEAD

A read only 32-bit integer value that indicates whether or not the connection is still active. CLI will return one of the following values:

- `SQL_CD_FALSE` - the connection is still active.
- `SQL_CD_TRUE` - an error has already happened and caused the connection to the server to be terminated. The application should still perform a disconnect to clean up any CLI resources.

## Connection attributes (CLI) list

This attribute is used mainly by the Microsoft ODBC Driver Manager 3.5x before pooling the connection.

### SQL\_ATTR\_CONNECTION\_TIMEOUT

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

### SQL\_ATTR\_CONNECTTYPE

A 32-bit integer value that specifies whether this application is to operate in a coordinated or uncoordinated distributed environment. The possible values are as follows:

- **SQL\_CONCURRENT\_TRANS (default):** The application can have concurrent multiple connections to any one database or to multiple databases. Each connection has its own commit scope. No effort is made to enforce coordination of transactions. If an application issues a commit using the environment handle on `SQLEndTran()` and not all of the connections commit successfully, the application is responsible for recovery.
- **SQL\_COORDINATED\_TRANS:** The application wishes to have commit and rollbacks coordinated among multiple database connections. This option setting corresponds to the specification of the Type 2 `CONNECT` in embedded SQL. In contrast to the `SQL_CONCURRENT_TRANS` setting described above, the application is permitted only one open connection per database.

**Note:** This connection type results in the default for `SQL_ATTR_AUTOCOMMIT` connection option to be `SQL_AUTOCOMMIT_OFF`.

If changing this attribute from the default then it must be set before any connections have been established on the environment handle.

It is recommended that the application set this attribute as an environment attribute with a call to `SQLSetEnvAttr()`, if necessary, as soon as the environment handle has been allocated. However, since ODBC applications cannot access `SQLSetEnvAttr()`, they must set this attribute using `SQLSetConnectAttr()` after each connection handle is allocated, but before any connections have been established.

All connections on an environment handle must have the same `SQL_ATTR_CONNECTTYPE` setting. An environment cannot have a mixture of concurrent and coordinated connections. The type of the first connection will determine the type of all subsequent connections. `SQLSetEnvAttr()` will return an error if an application attempts to change the connection type while there is an active connection.

The default connect type can also be set using the "ConnectType CLI/ODBC configuration keyword" on page 343.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_CURRENT\_CATALOG

A null-terminated character string containing the name of the catalog used by the data source. The catalog name is typically the same as the database name.



This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`. Any attempt to set this attribute will result in an `SQLSTATE` of `HYC00` (Driver not capable).

### SQL\_ATTR\_CURRENT\_IMPLICIT\_XMLPARSE\_OPTION

A null-terminated character string that is the string constant used to set the `CURRENT IMPLICIT XMLPARSE OPTION` special register. Setting this attribute causes the `SET CURRENT IMPLICIT XMLPARSE OPTION SQL` statement to be issued. If this attribute is set before a connection has been established, the `SET CURRENT IMPLICIT XMLPARSE OPTION SQL` statement will be issued when the connection is made.

This attribute is not supported when accessing Informix database server.

### SQL\_ATTR\_CURRENT\_PACKAGE\_PATH

A null-terminated character string of package qualifiers that the DB2 database server uses to try to resolve the package when multiple packages have been configured. Setting this attribute causes the `"SET CURRENT PACKAGE PATH = schema1, schema2, ..."` statement to be issued after every connection to the database server.

This attribute is best suited for use with ODBC static processing applications, rather than CLI applications.

This attribute is not supported when accessing Informix database server.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_CURRENT\_PACKAGE\_SET

A null-terminated character string that indicates the schema name (collection identifier) that is used to select the package for subsequent `SQL` statements. Setting this attribute causes the `SET CURRENT PACKAGESET SQL` statement to be issued. If this attribute is set before a connection, the `SET CURRENT PACKAGESET SQL` statement will be issued at connection time.

CLI/ODBC applications issue dynamic `SQL` statements. Using this connection attribute, you can control the privileges used to run these statements:

- Choose a schema to use when running `SQL` statements from CLI/ODBC applications.
- Ensure the objects in the schema have the desired privileges and then rebind accordingly. This typically means binding the CLI packages (`sqllib/bnd/db2cli.lst`) using the `COLLECTION <collid>` option. Refer to the `BIND` command for further details.
- Set the `CURRENTPACKAGESET` option to this schema.

The `SQL` statements from the CLI/ODBC applications will now run under the specified schema and use the privileges defined there.

Setting the `"CurrentPackageSet CLI/ODBC configuration keyword"` on page 346 is an alternative method of specifying the schema name.

The following package set names are reserved: `NULLID`, `NULLIDR1`, `NULLIDRA`.

`SQL_ATTR_REOPT` and `SQL_ATTR_CURRENT_PACKAGE_SET` are mutually exclusive, therefore, if one is set, the other is not allowed.

This attribute is not supported when accessing Informix database server.

## Connection attributes (CLI) list

### SQL\_ATTR\_CURRENT\_SCHEMA

A null-terminated character string containing the name of the schema to be used by CLI for the `SQLColumns()` call if the `szSchemaName` pointer is set to null.

To reset this option, specify this option with a zero length string or a null pointer for the `ValuePtr` argument.

This option is useful when the application developer has coded a generic call to `SQLColumns()` that does not restrict the result set by schema name, but needs to constrain the result set at isolated places in the code.

This option can be set at any time and will be effective on the next `SQLColumns()` call where the `szSchemaName` pointer is null.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_DB2\_APPLICATION\_HANDLE

A user-defined character string that returns the application handle of the connection. If the string is not large enough to contain the complete application handle, it will be truncated.

This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`.

This attribute is not supported when accessing Informix database server.

### SQL\_ATTR\_DB2\_APPLICATION\_ID

A user-defined character string that returns the application identifier of the connection. If the string is not large enough to contain the complete application identifier, it will be truncated.

This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`.

This attribute is not supported when accessing Informix database server.

### SQL\_ATTR\_DB2\_SQLERRP

A sqlpointer to a null-terminated string containing the `sqlerrp` field of the `sqlca`.

Begins with a three-letter identifier indicating the product, followed by five alphanumeric characters indicating the version, release, and modification level of the product. The characters A-Z indicate a modification level higher than 9. A indicates modification level 10, B indicates modification level 11, and so on. For example, `SQL0907C` means DB2 Version 9 Release 7 Modification level 12.

If `SQLCODE` indicates an error condition, then this field identifies the module that returned the error.

This field is also used when a successful connection is completed.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_DB2ESTIMATE

This attribute has been deprecated in DB2 UDB Version 8.

### SQL\_ATTR\_DB2EXPLAIN

A 32-bit integer that specifies whether Explain snapshot, Explain mode information, or both should be generated by the server. Permitted values are:

- `SQL_DB2EXPLAIN_OFF`: Both the Explain Snapshot and the Explain table option facilities are disabled (a `SET CURRENT EXPLAIN SNAPSHOT=NO` and a `SET CURRENT EXPLAIN MODE=NO` are sent to the server).
- `SQL_DB2EXPLAIN_SNAPSHOT_ON`: The Explain Snapshot facility is enabled, and the Explain table option facility is disabled (a `SET CURRENT EXPLAIN SNAPSHOT=YES` and a `SET CURRENT EXPLAIN MODE=NO` are sent to the server).
- `SQL_DB2EXPLAIN_MODE_ON`: The Explain Snapshot facility is disabled, and the Explain table option facility is enabled (a `SET CURRENT EXPLAIN SNAPSHOT=NO` and a `SET CURRENT EXPLAIN MODE=YES` are sent to the server).
- `SQL_DB2EXPLAIN_SNAPSHOT_MODE_ON`: Both the Explain Snapshot and the Explain table option facilities are enabled (a `SET CURRENT EXPLAIN SNAPSHOT=YES` and a `SET CURRENT EXPLAIN MODE=YES` are sent to the server).

Before the explain information can be generated, the explain tables must be created.

This statement is not under transaction control and is not affected by a `ROLLBACK`. The new `SQL_ATTR_DB2EXPLAIN` setting is effective on the next statement preparation for this connection.

The current authorization ID must have `INSERT` privilege for the Explain tables.

The default value can also be set using the “DB2Explain CLI/ODBC configuration keyword” on page 349.

This attribute is not supported when accessing Informix database server.

**Note:** This is an IBM defined extension.

### `SQL_ATTR_DECFLOAT_ROUNDING_MODE`

The decimal float rounding mode determines what type of rounding will be used if a value is put into a `DECFLOAT` variable or column but the value has more digits than are allowed in the `DECFLOAT` data type. This can occur when inserting, updating, selecting, converting from another type, or as the result of a mathematical operation.

The value of `SQL_ATTR_DECFLOAT_ROUNDING_MODE` determines the decimal float rounding mode that will be used for new connections unless another mode is specified by a connection attribute for that connection. For any given connection both CLI and DB2 will use the same decimal float rounding mode for all action initiated as part of that connection.

When your applications are connecting to a DB2 Database for Linux, UNIX, and Windows Version 9.5 server, you must set the decimal float rounding mode on the database client to the same mode that is set on the server. If you set the decimal float rounding mode on the client to a value that is different from the decimal float rounding mode that is set on the database server, the database server will return `SQL0713N` on connection.

The settings correspond to these decimal float rounding modes:

- 0 = Half even (default)
- 1 = Half up
- 2 = Down

## Connection attributes (CLI) list

- 3 = Ceiling
- 4 = Floor

The different modes are:

### Half even (default)

In this mode CLI and DB2 use the number that will fit in the target variable and that is closest to the original value. If two numbers are equally close, they use the one that is even. This mode produces the smallest rounding errors over large amounts of data.

### Half up

In this mode CLI and DB2 use the number that will fit in the target variable and that is closest to the original value. If two numbers are equally close, they use the one that is greater than the original value.

**Down** In this mode CLI and DB2 use the number that will fit in the target variable and that is closest to the original value and for which the absolute value is not greater than the absolute value of the original value. You can also think of this as rounding toward zero or as using ceiling for negative values and using floor for positive values.

### Ceiling

In this mode CLI and DB2 use the smallest number that will fit in the target variable and that is greater than or equal to the original value.

**Floor** In this mode CLI and DB2 use the largest number that will fit in the target variable and that is less than or equal to the original value.

This attribute is not supported when accessing IDS data servers.

## SQL\_ATTR\_DESCRIBE\_CALL

A 32-bit integer value that indicates when stored procedure arguments are described. By default, CLI does not request input parameter describe information when it prepares a CALL statement. If an application has correctly bound parameters to a statement, then this describe information is unnecessary and not requesting it improves performance. The option values are:

- 1 = SQL\_DESCRIBE\_CALL\_BEFORE.
- -1 = SQL\_DESCRIBE\_CALL\_DEFAULT.

Setting this attribute can be done using the “DescribeCall CLI/ODBC configuration keyword” on page 355. Refer to the keyword for usage information and descriptions of the available options.

**Note:** This is an IBM defined extension.

## SQL\_ATTR\_DESCRIBE\_OUTPUT\_LEVEL

A null-terminated character string that controls the amount of information the CLI driver requests on a prepare or describe request. By default, when the server receives a describe request, it returns the information contained in level 2 of Table 167 on page 450 for the result set columns. An application, however, might not need all of this information or might need additional information. Setting the SQL\_ATTR\_DESCRIBE\_OUTPUT\_LEVEL attribute to a level that suits the needs of the client application might improve performance because the

describe data transferred between the client and server is limited to the minimum amount that the application requires. If the `SQL_ATTR_DESCRIBE_OUTPUT_LEVEL` setting is set too low, it might impact the functionality of the application (depending on the application's requirements). The CLI functions to retrieve the describe information might not fail in this case, but the information returned might be incomplete. Supported settings for `SQL_ATTR_DESCRIBE_OUTPUT_LEVEL` are:

- 0 - no describe information is returned to the client application
- 1 - describe information categorized in level 1 (see Table 167 on page 450) is returned to the client application
- 2 - (default) describe information categorized in level 2 (see Table 167 on page 450) is returned to the client application
- 3 - describe information categorized in level 3 (see Table 167 on page 450) is returned to the client application

The following table lists the fields that form the describe information that the server returns when it receives a prepare or describe request. These fields are grouped into levels, and the `SQL_ATTR_DESCRIBE_OUTPUT_LEVEL` attribute controls which levels of describe information the CLI driver requests.

**Note:**

1. Not all levels of describe information are supported by all DB2 servers. All levels of describe information are supported on the following DB2 servers: DB2 on Linux, UNIX, and Windows Version 8 and later, DB2 for z/OS Version 8 and later, and DB2 for i Version 5 Release 3 and later. All other DB2 servers support only the 2 or 0 setting for `SQL_ATTR_DESCRIBE_OUTPUT_LEVEL`.
2. The default behavior will allow CLI to promote the level to 3 if the application asks for describe information that was not initially retrieved using the default level 2. This might result in two network flows to the server. If an application uses this attribute to explicitly set a describe level, then no promotion will occur. Therefore, if the attribute is used to set the describe level to 2, then CLI will not promote to level 3 even if the application asks for extended information.

## Connection attributes (CLI) list

Table 167. Levels of describe information

Level 1	Level 2	Level 3
SQL_DESC_COUNT	all fields of level 1 and: SQL_DESC_NAME SQL_DESC_LABEL SQL_COLUMN_NAME SQL_DESC_UNNAMED SQL_DESC_TYPE_NAME SQL_DESC_DISTINCT_TYPE SQL_DESC_REFERENCE_TYPE SQL_DESC_STRUCTURED_TYPE SQL_DESC_USER_TYPE SQL_DESC_LOCAL_TYPE_NAME SQL_DESC_USER_DEFINED_ TYPE_CODE	all fields of levels 1 and 2 and: SQL_DESC_BASE_COLUMN_NAME SQL_DESC_UPDATABLE SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_SCHEMA_NAME SQL_DESC_CATALOG_NAME SQL_DESC_TABLE_NAME SQL_DESC_BASE_TABLE_NAME
SQL_COLUMN_COUNT		
SQL_DESC_TYPE		
SQL_DESC_CONCISE_TYPE		
SQL_COLUMN_LENGTH		
SQL_DESC_OCTET_LENGTH		
SQL_DESC_LENGTH		
SQL_DESC_PRECISION		
SQL_COLUMN_PRECISION		
SQL_DESC_SCALE		
SQL_COLUMN_SCALE		
SQL_DESC_DISPLAY_SIZE		
SQL_DESC_NULLABLE		
SQL_COLUMN_NULLABLE		
SQL_DESC_UNSIGNED		
SQL_DESC_SEARCHABLE		
SQL_DESC_LITERAL_SUFFIX		
SQL_DESC_LITERAL_PREFIX		
SQL_DESC_CASE_SENSITIVE		
SQL_DESC_FIXED_PREC_SCALE		

Setting the “DescribeOutputLevel CLI/ODBC configuration keyword” on page 356 is an alternative method of specifying this behavior.

### SQL\_ATTR\_ENLIST\_IN\_DTC

An SQLPOINTER which can be either of the following:

- non-null transaction pointer: The application requests to CLI to change the state of the connection from non-distributed transaction state to distributed state. The connection is enlisted with the Distributed Transaction Coordinator (DTC).
- null: The application requests to CLI to change the state of the connection from distributed transaction state to a non-distributed transaction state.

This attribute is only used in a Microsoft Transaction Server (MTS) environment to enlist or un-enlist a connection with MTS.

Each time this attribute is used with a non-null transaction pointer, the previous transaction is assumed to be ended and a new transaction is initiated. The application must call the ITransaction member function Endtransaction before calling this API with a non-null pointer. Otherwise the previous transaction will be aborted. The application can enlist multiple connections with the same transaction pointer.

**Note:** This connection attribute is specified by MTS automatically for each transaction and is not coded by the user application.

It is imperative for CLI/ODBC applications that there will be no concurrent SQL statements executing on 2 different connections into the same database that are enlisted in the same transaction.

### SQL\_ATTR\_EXTENDED\_INDICATORS

A 32-bit integer that allows users to use the extended indicator feature from the supported server. If the user attempts to set this attribute against the data server which does not support extended indicators, an appropriate error is returned to the CLI application. This attribute can take the following value:

- `SQL_EXTENDED_INDICATOR_ENABLE`: Enables users to specify values to signify `SQL_UNASSIGNED` and `SQL_DEFAULT_PARAM` on the `SQLBindParameter()` / `SQLExtendedBind()` methods.
- `SQL_EXTENDED_INDICATOR_NOT_SET` (default): This feature is disabled by default. The user gets an `InvalidArgument` value error, `CLI0124E`, if the `SQL_UNASSIGNED` and `SQL_DEFAULT_PARAM` are used before enabling this feature using `SQL_ATTR_EXTENDED_INDICATORS`.
- Extended indicators support DB2 for Linux, UNIX, and Windows and for DB2 10 for z/OS data servers starts in DB2 Version 9.7 Fix Pack 2. Extended indicators support DB2 for IBM i 7.1 data servers starts in DB2 Version 9.7 Fix Pack 5.

### `SQL_ATTR_FET_BUF_SIZE`

A connection level attribute to allow applications to set the default query block size to an optimum value in range of 64K-256K. This attribute should be set before a connection is made. CLI will also provide a `db2cli.ini` level keyword, `FET_BUF_SIZE`, which can be set in `db2cli.ini` file and connection string.

An equivalent `db2dsdriver.cfg` keyword, `FetchBufferSize` is also available, which can be set in the `db2dsdriver.cfg` file.

### `SQL_ATTR_FREE_LOCATORS_ON_FETCH`

A boolean attribute that specifies if LOB locators are freed when `SQLFetch()` is executed, rather than when a `COMMIT` is issued. Setting this attribute to 1 (true) frees the locators that are used internally when applications fetch LOB data without binding the LOB columns with `SQLBindCol()` (or equivalent descriptor APIs). Locators that are explicitly returned to the application must still be freed by the application. This attribute value can be used to avoid scenarios where an application receives `SQLCODE = -429` (no more locators). The default for this attribute is 0 (false).

**Note:** This is an IBM defined extension.

### `SQL_ATTR_FORCE_ROLLBACK`

A 32-bit unsigned integer value that allows calls to the `SQLEndTran()` function in a data-at-execution flow for connections to DB2 for z/OS and OS/390 servers.

To call the `SQLEndTran()` function specifying `SQL_ROLLBACK` as *CompletionType* in your applications during a data-at-execution flow, the `StreamPutData` configuration keyword must be set to 1, and the `SQL_ATTR_FORCE_ROLLBACK` connection attribute must also be set.

The `CLI0150E` error message is returned for connections to data servers that are not DB2 for z/OS and OS/390 servers.

**Note:** This is an IBM defined extension.

### `SQL_ATTR_GET_LATEST_MEMBER`

A connection level attribute, which is available starting in Version 9.7 Fix Pack 3, enables CLI applications to retrieve the latest member (being) used for the given connection.

CLI applications can retrieve the currently connected or last connected member on a connection, by calling the `SQLGetConnectAttr()` function.

## Connection attributes (CLI) list

The CLI0126E error message is returned if you try to use this attribute before establishing a database connection.

### SQL\_ATTR\_INFO\_ACCTSTR

A pointer to a null-terminated character string used to identify the client accounting string sent to the data server when using DB2 Connect or DB2 Database for Linux, UNIX, and Windows.

Note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 200 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the SQL\_ATTR\_INFO\_ACCTSTR attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 - 9, and the underscore (\_) or period (.)

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_INFO\_APPLNAME

A pointer to a null-terminated character string used to identify the client application name sent to the data server when using DB2 Connect or DB2 database products for Linux, UNIX and Windows.

Note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 32 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the SQL\_ATTR\_INFO\_APPLNAME attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 - 9, and the underscore (\_) or period (.)

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_INFO\_PROGRAMID

A user-defined character string, with a maximum length of 80 bytes, that associates an application with a connection. Once this attribute is set, DB2 for z/OS Version 8 and later associates this attribute with any statements inserted into the dynamic SQL statement cache.

This attribute is supported for CLI applications accessing DB2 for z/OS Version 8 and later or IBM Informix database server.

Also, in DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the SQL\_ATTR\_INFO\_PROGRAMID attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.

**Note:** This is an IBM defined extension.



### SQL\_ATTR\_INFO\_PROGRAMNAME

A null-terminated user-defined character string, up to 20 bytes in length, used to specify the name of the application running on the client.

When this attribute is set before the connection to the server is established, the value specified overrides the actual client application name and will be the value that is displayed in the `appl_name` monitor element. When connecting to a DB2 for z/OS server, the first 12 characters of this setting are used as the `CORRELATION IDENTIFIER` of the associated DB2 for z/OS thread.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_INFO\_USERID

A pointer to a null-terminated character string used to identify the client user ID sent to the data server when using DB2 Connect or DB2 database products for Linux, UNIX and Windows.

Note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 16 characters.
- This user ID is not to be confused with the authentication user ID. This user ID is for identification purposes only and is not used for any authorization.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the `SQL_ATTR_INFO_USERID` attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 - 9, and the underscore (`_`) or period (`.`).

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_INFO\_WRKSTNNAME

A pointer to a null-terminated character string used to identify the client workstation name sent to the data server when using DB2 Connect or DB2 database products for Linux, UNIX and Windows. In Version 9.7 Fix pack 6 and later, if `SQL_ATTR_INFO_WRKSTNNAME` attribute is not specified, default value that consists of the host name is used. The host name is obtained by `gethostname()` function call. If host name is not configured or an error is encountered during a `gethostname()` function call, no value for the `SQL_ATTR_INFO_WRKSTNNAME` attribute is sent to the server.

Note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 18 characters.
- In DB2 Version 9.7 Fix Pack 6 and later, CLI applications can set the `SQL_ATTR_INFO_WRKSTNNAME` attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 - 9, and the underscore (`_`) or period (`.`).

## Connection attributes (CLI) list

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_KEEP\_DYNAMIC

A 32-bit unsigned integer value which specifies whether the KEEP\_DYNAMIC option has been enabled. If enabled, the server will keep dynamically prepared statements in a prepared state across transaction boundaries.

- 0 - KEEP\_DYNAMIC functionality is not available; CLI packages were bound with the KEEP\_DYNAMIC NO option
- 1 - KEEP\_DYNAMIC functionality is available; CLI packages were bound with the KEEP\_DYNAMIC YES option

It is recommended that when this attribute is set, the SQL\_ATTR\_CURRENT\_PACKAGE\_SET attribute also be set.

This attribute is not supported when accessing Informix database server.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_LOB\_CACHE\_SIZE

A 32-bit unsigned integer that specifies maximum cache size (in bytes) for LOBs. By default, LOBs are not cached.

See the "LOBCacheSize CLI/ODBC configuration keyword" on page 366 for further usage information.

### SQL\_ATTR\_LOGIN\_TIMEOUT

A 32-bit integer value corresponding to the number of seconds to wait for a reply when trying to establish a connection to a server before terminating the attempt and generating a communication timeout. Specify a positive integer, up to 32 767. The default setting of 0 will allow the client to wait indefinitely.

Setting a connection timeout value can also be done using the "ConnectTimeout CLI/ODBC configuration keyword" on page 342. Refer to the keyword for usage information.

### SQL\_ATTR\_LONGDATA\_COMPAT

A 32-bit integer value indicating whether the character, double byte character and binary large object data types should be reported respectively as SQL\_LONGVARCHAR, SQL\_LONGVARGRAPHIC or SQL\_LONGBINARY, enabling existing applications to access large object data types seamlessly. The option values are:

- **SQL\_LD\_COMPAT\_NO (default):** The large object data types are reported as their respective IBM-defined types (SQL\_BLOB, SQL\_CLOB, SQL\_DBCLOB).
- **SQL\_LD\_COMPAT\_YES:** The IBM large object data types (SQL\_BLOB, SQL\_CLOB and SQL\_DBCLOB) are mapped to SQL\_LONGVARBINARY, SQL\_LONGVARCHAR and SQL\_LONGVARGRAPHIC; SQLGetTypeInfo() returns one entry each for SQL\_LONGVARBINARY, SQL\_LONGVARCHAR, and SQL\_LONGVARGRAPHIC.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_MAPCHAR

A 32-bit integer value used to specify the default SQL type associated with SQL\_CHAR, SQL\_VARCHAR, SQL\_LONGVARCHAR. The option values are:

- **SQL\_MAPCHAR\_DEFAULT (default):** return the default SQL type representation
- **SQL\_MAPCHAR\_WCHAR:** return SQL\_CHAR as SQL\_WCHAR, SQL\_VARCHAR as SQL\_WVARCHAR, and SQL\_LONGVARCHAR as SQL\_WLONGVARCHAR

Only the following CLI functions are affected by setting this attribute:

- SQLColumns()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()

Setting the default SQL type associated with SQL\_CHAR, SQL\_VARCHAR, SQL\_LONGVARCHAR can also be done using the “MapCharToWChar CLI/ODBC configuration keyword” on page 369.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_MAXCONN

This attribute has been deprecated in DB2 UDB Version 8.

### SQL\_ATTR\_MAX\_LOB\_BLOCK\_SIZE

A 32-bit unsigned integer that indicates the maximum size of LOB or XML data block. Specify a positive integer, up to 2 147 483 647. The default setting of 0 indicates that there is no limit to the data block size for LOB or XML data.

During data retrieval, the server will include all of the information for the current row in its reply to the client even if the maximum block size has been reached.

If both MaxLOBBlockSize and the db2set registry variable DB2\_MAX\_LOB\_BLOCK\_SIZE are specified, the value for MaxLOBBlockSize will be used.

Setting the “MaxLOBBlockSize CLI/ODBC configuration keyword” on page 376 is an alternative method of specifying this behavior.

### SQL\_ATTR\_METADATA\_ID

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

### SQL\_ATTR\_NETWORK\_STATISTICS

Starting in Version 9.7 Fix Pack 3, this 32-bit integer connection attribute controls whether CLI collects network statistics for a connection. An application can retrieve the network statistics for a connection by calling the SQLGetDiagField() function and specifying SQL\_DIAG\_NETWORK\_STATISTICS for the *DiagIdentifier* argument.

In Version 9.7 Fix pack 6 and later, you can obtain the server time for COMMIT or ROLLBACK SQL operations on DB2 for z/OS Version 10 and later by calling the SQLGetDiagField() function and specifying SQL\_DIAG\_NETWORK\_STATISTICS for *DiagIdentifier* argument after the SQLEndTran() function call. You must enable SQL\_ATTR\_NETWORK\_STATISTICS attribute and DB2 for z/OS Version

## Connection attributes (CLI) list

10 and later server must have fix for APAR PM53243 to obtain server time for COMMIT or ROLLBACK SQL operations.

The permitted values are as follows:

### **SQL\_NETWORK\_STATISTICS\_OFF (default)**

Disables network statistics collection for a connection.

### **SQL\_NETWORK\_STATISTICS\_ON**

Enables network statistics collection for a connection.

### **SQL\_NETWORK\_STATISTICS\_ON\_SKIP\_NOSERVER**

In addition to enabling network statistics collection for a connection, network flows are omitted that are known to have no server time reported, for example explicit COMMIT and ROLLBACK statements.

Requests that have no server time reported can affect the usefulness of returned information, if calculations are made that subtract the server time from the network time. The `SQL_NETWORK_STATISTICS_ON_SKIP_NOSERVER` option excludes these requests from the values reported. Only explicit, unchained requests are excluded; autocommit and chained COMMIT statements are not skipped.

Starting in Version 9.7 Fix Pack 5, CLI collects statistics for server time reported on COMMIT and ROLLBACK. The DB2 server must be at a level that supports reporting server time for COMMIT and ROLLBACK.

### **SQL\_ATTR\_ODBC\_CURSORS**

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

### **SQL\_ATTR\_OVERRIDE\_CODEPAGE**

This connection attribute, which is available starting in Version 9.7 Fix Pack 5, enables CLI applications to fetch or insert data of CHARACTER or GRAPHIC data type without code page conversions. The possible values are as follows:

- `SQL_OVERRIDE_CODEPAGE_ON`: CLI does not perform codepage conversions for binding of character or graphic data.
- **`SQL_OVERRIDE_CODEPAGE_OFF` (default)**: CLI performs codepage conversions for binding of character or graphic data.

If you set this attribute to `SQL_OVERRIDE_CODEPAGE_ON`, you must ensure that data is in the correct code page.

If you set `SQL_ATTR_OVERRIDE_CODEPAGE` to `SQL_OVERRIDE_CODEPAGE_ON` after setting `SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE`, CLI returns the CLI0126E error message.

### **SQL\_ATTR\_OVERRIDE\_CHARACTER\_CODEPAGE**

This connection attribute, which is available starting in Version 9.7 Fix Pack 3, enables CLI applications to specify the database code page. The code page does not have to be available at the client end.

If you specify the same code page as the database code page, applications can fetch or insert data of CHARACTER data type without any code page conversions.

Setting this attribute after allocating statement handles results in error CLI0126E (invalid operation). Setting the attribute to a value that is not supported by the database results in error CLI0210E (inconsistent code page value error).

If you set `SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE` after setting `SQL_ATTR_OVERRIDE_CODEPAGE` to `SQL_OVERRIDE_CODEPAGE_ON`, CLI returns the CLI0126E error message.

During a bind-out operation, ensure that the CLI applications allocate buffers large enough to hold the retrieved data during bind-out operations. If there is insufficient space, error CLI0002W is returned.

**Restriction:** This attribute is supported only for DB2 for z/OS data servers. If you attempt to set the value of this attribute for other data servers, error CLI0150E (driver not capable) is returned.

### SQL\_ATTR\_PACKET\_SIZE

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

### SQL\_ATTR\_PARC\_BATCH

For applications that use array input to achieve bulk inserts, deletes, or updates, this 32-bit unsigned integer connection attribute indicates whether the application receives the number of rows in a table that were affected by the each parameter set or the cumulative number of rows that were affected for the entire parameter set. This connection attribute is available starting in DB2 Version 9.7 Fix Pack 5 for non-atomic operations. The possible values are as follows:

- `SQL_PARC_BATCH_ENABLE`: CLI returns the number of rows in a table that were affected by the each parameter set.
- **SQL\_PARC\_BATCH\_DISABLE (default)**: CLI returns the total number of rows that were affected for the entire parameter set.

If you set this connection attribute to `SQL_PARCH_BATCH_ENABLE`, you must indicate an array as the *RowCountPtr* parameter in the `SQLRowCount()` function and you must set `SQL_ATTR_PARAMOPT_ATOMIC` to `SQL_ATOMIC_NO`. If `SQL_ATTR_PARAMOPT_ATOMIC` is set to `SQL_ATOMIC_YES`, the CLI0150E error message is returned when you call the `SQLExecute()` function.

### SQL\_ATTR\_PING\_DB

A 32-bit integer which is used with `SQLGetConnectAttr()` to get the ping time in microseconds.

If a connection has previously been established and has been dropped by the database, a value of 0 is reported. If the connection has been closed by the application, then an SQLSTATE of 08003 is reported. This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`. Any attempt to set this attribute will result in an SQLSTATE of 7HYC00 (Driver not capable)

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_PING\_NTIMES

A 32-bit integer that is used with `SQLGetConnectAttr()` that sets the number of ping iterations that CLI performs when the application uses

## Connection attributes (CLI) list

`SQL_ATTR_PING_DB`. If you set `SQL_ATTR_PING_NTIMES` to a value greater than 1, `SQL_ATTR_PING_DB` returns the average time that CLI took to ping the database for the set of iterations.

This attribute has a valid range from 1 to 32767 (inclusive). `SQLGetConnectAttr()` checks the value and returns the appropriate error code when the value is outside this range.

### `SQL_ATTR_PING_REQUEST_PACKET_SIZE`

A 32-bit integer that is used with `SQLGetConnectAttr()` that sets the size of the ping packet that CLI uses when the application uses `SQL_ATTR_PING_DB`.

This attribute has a valid range from 1 to 32767 (inclusive). `SQLGetConnectAttr()` checks the value and returns the appropriate error code when the value is outside this range.

### `SQL_ATTR_QUIET_MODE`

A 32-bit platform specific window handle.

If the application has never made a call to `SQLSetConnectAttr()` with this option, then CLI would return a null parent window handle on `SQLGetConnectAttr()` for this option and use a null parent window handle to display dialogue boxes. For example, if the end user has asked for (via an entry in the CLI initialization file) optimizer information to be displayed, CLI would display the dialogue box containing this information using a null window handle. (For some platforms, this means the dialogue box would be centered in the middle of the screen.)

If *ValuePtr* is set to null, then CLI does not display any dialogue boxes. In the above example where the end user has asked for the optimizer estimates to be displayed, CLI would not display these estimates because the application explicitly wants to suppress all such dialogue boxes.

If *ValuePtr* is not null, then it should be the parent window handle of the application. CLI uses this handle to display dialogue boxes. (For some platforms, this means the dialogue box would be centered with respect to the active window of the application.)

**Note:** This connection option cannot be used to suppress the `SQLDriverConnect()` dialogue box (which can be suppressed by setting the *fDriverCompletion* argument to `SQL_DRIVER_NOPROMPT`).

### `SQL_ATTR_RECEIVE_TIMEOUT`

A 32-bit integer value that is the number of seconds a client waits for a reply from a server on an established connection before terminating the attempt and generating a communication timeout error. The default value of 0 indicates the client waits indefinitely for a reply. The receive timeout has no effect during connection establishment; it is only supported for TCP/IP, and is ignored for any other protocol. Supported values are integers from 0 to 32767.

**Note:** This is an IBM defined extension.

### `SQL_ATTR_REOPT`

A 32-bit integer value that enables query optimization for SQL statements that contain special registers or parameter markers. Optimization occurs by using the values available at query execution time for special registers or

parameter markers, instead of the default estimates that are chosen by the compiler. The valid values of the attribute are:

- **2 = SQL\_REOPT\_NONE (default):** No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for the special registers or parameter markers. The default NULLID package set is used to execute dynamic SQL statements.
- **3 = SQL\_REOPT\_ONCE:** Query optimization occurs once at query execution time, when the query is executed for the first time. The NULLIDR1 package set, which is bound with the REOPT ONCE bind option, is used.
- **4 = SQL\_REOPT\_ALWAYS:** Query optimization or reoptimization occurs at query execution time every time the query is executed. The NULLIDRA package set, which is bound with the REOPT ALWAYS bind option, is used.

The NULLIDR1 and NULLIDRA are reserved package set names, and when used, REOPT ONCE and REOPT ALWAYS are implied respectively. These package sets have to be explicitly created with these commands:

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

SQL\_ATTR\_REOPT and SQL\_ATTR\_CURRENT\_PACKAGE\_SET are mutually exclusive, therefore, if one is set, the other is not allowed.

This attribute is not supported when accessing Informix database server.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_REPORT\_ISLONG\_FOR\_LONGTYPES\_OLEDB

A 32-bit integer value. The OLE DB client cursor engine and the OLE DB .NET Data Provider CommandBuilder object generate UPDATE and DELETE statements based on column information provided by the IBM DB2 OLE DB Provider. If the generated statement contains a LONG type in the WHERE clause, the statement will fail because LONG types cannot be used in a search with an equality operator. The possible values are as follows:

- **0 (default):** LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) do not have the DBCOLUMNFLAGS\_ISLONG flag set, which might cause the columns to be used in the WHERE clause.
- **1:** The IBM DB2 OLE DB Provider reports LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) with the DBCOLUMNFLAGS\_ISLONG flag set. This will prevent the long columns from being used in the WHERE clause.

This attribute is supported by the following database servers:

- DB2 for z/OS
  - Version 6 with PTF UQ93891
  - Version 7 with PTF UQ93889
  - Version 8 with PTF UQ93890
  - Versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
  - Version 8.2 (equivalent to Version 8.1, Fix Pack 7) and later

## Connection attributes (CLI) list

This attribute is not supported when accessing Informix database server.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_REPORT\_SEAMLESSFAILOVER\_WARNING

A 32-bit unsigned integer value that specifies whether to return a warning message on execute requests if a seamless failover occurred during the request. This connection attribute is available starting in DB2 Version 9.7 Fix Pack 5. The possible values are as follows:

- **SQL\_REPORT\_SEAMLESSFAILOVER\_WARNING\_YES:** If a seamless failover occurred during an execute request, a warning message is returned.
- **SQL\_REPORT\_SEAMLESSFAILOVER\_WARNING\_NO (default):** If a seamless failover occurred during an execute request, a warning message is not returned.

### SQL\_ATTR\_REPORT\_TIMESTAMP\_TRUNC\_AS\_WARN

A 32-bit unsigned integer value that specifies whether a datetime overflow results in an error (SQLSTATE 22008) or warning (SQLSTATE 01S07). The possible values are as follows:

- **0 (default):** Datetime overflow results in an error (SQLSTATE 22008).
- **1:** Datetime overflow results in a warning (SQLSTATE 01S07).

### SQL\_ATTR\_RETRYONERROR

CLI attempts to recover from non-fatal errors, such as incorrect binding of application parameters, by retrieving additional information on the failing SQL statement and then executing the statement again. The additional information retrieved includes input parameter information from the database catalog tables. If CLI is able to recover successfully from the error, by default, it does not report the error to the application. The CLI/ODBC configuration keyword ReportRetryErrorsAsWarnings allows you to set whether error recovery warnings are returned to the application or not.

**Note:** Once CLI has successfully completed the error recovery, the application may behave differently, because CLI uses the catalog information gathered during the recovery for subsequent executions of that particular SQL statement, rather than the information provided in the original SQLBindParameter() function calls. If you do not want this behavior, set RetryOnError to 0, forcing CLI not to attempt recovery. You should, however, modify the application to correctly bind statement parameters.

### SQL\_ATTR\_SERVER\_MSGTXT\_MASK

A 32-bit integer value used to indicate when CLI should request the error message from the server. This attribute is used in conjunction with the SQL\_ATTR\_SERVER\_MSGTXT\_SP attribute. The attribute can be set to:

- **SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_LOCAL\_FIRST (default):** CLI will check the local message files first to see if the message can be retrieved. If no matching SQLCODE is found, then CLI will request the information from the server.
- **SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_WARNINGS:** CLI always requests the message information from the server for warnings but error messages are retrieved from the local message files.
- **SQL\_ATTR\_SERVER\_MSGTXT\_MASK\_ERRORS:** CLI always requests the message information from the server for errors but warning messages are retrieved from the local message files.



- `SQL_ATTR_SERVER_MSGTXT_MASK_ALL`: CLI always requests the message information from the server for both error and warning messages.

Setting the “ServerMsgMask CLI/ODBC configuration keyword” on page 397 is an alternative method of specifying this behavior.

**Note:** This is an IBM defined extension.

### `SQL_ATTR_SERVER_MSGTXT_SP`

A pointer to a character string used to identify a stored procedure that is used for generating an error message based on an SQLCA. This can be useful when retrieving error information from a server such as DB2 for z/OS. The attribute can be set to:

- `SYSIBM.SQLCAMESSAGE`: The default procedure called to retrieve the message text from DB2 for z/OS servers. If you do not set this attribute, this procedure is called.
- `DSNACCMG`: The default procedure called to retrieve the message text from DB2 for z/OS Version 7 servers. The `SYSIBM.SQLCAMESSAGE` procedure is called to retrieve the message text from DB2 for z/OS Version 8 or later. `DSNACCMG` has been deprecated in DB2 for z/OS Version 9 and might be removed in a future release.
- Any user-created stored procedure.

Applications using this attribute can also set the `SQL_ATTR_SERVER_MSGTXT_MASK` attribute to indicate when CLI should call this procedure to retrieve the message information from the server. If the `SQL_ATTR_SERVER_MSGTXT_MASK` is not set, then the default is to check the local message files first (see `SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST` in `SQL_ATTR_SERVER_MSGTXT_MASK`).

Setting the “UseServerMsgSP CLI/ODBC configuration keyword” on page 423 is an alternative method of specifying this behavior.

**Note:** This is an IBM defined extension.

### `SQL_ATTR_SESSION_TIME_ZONE`

A null-terminated character string in the format  $\pm hh:mm$ , containing the server session time zone information. This is a set-only attribute. The supported time zone values range from -12:59 through +14:00.

### `SQL_ATTR_SQLCODEMAP`

Specifies whether SQLCODE mapping should be set to default or turned off. The possible values are as follows:

- **MAP (default)**: SQLCODE mapping is set.
- **NOMAP**: SQLCODEMapping is turned off.

### `SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB`

A 32-bit integer value. The Microsoft OLE DB specification requires that `IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS)` returns the row set sorted by the columns `TABLE_CATALOG`, `TABLE_SCHEMA`, `TABLE_NAME`, `COLUMN_NAME`. The IBM DB2 OLE DB Provider conforms to the specification, however, applications that use the Microsoft ODBC Bridge provider (MSDASQL) have been typically coded to get the row set sorted by `ORDINAL_POSITION`. The possible values are as follows:

## Connection attributes (CLI) list

- **0 (default):** The IBM DB2 OLE DB Provider returns a row set sorted by the columns TABLE\_CATALOG, TABLE\_SCHEMA, TABLE\_NAME, COLUMN\_NAME.
- **1:** The IBM DB2 OLE DB Provider returns a row set sorted by ORDINAL\_POSITION.

This attribute is supported by the following database servers:

- DB2 for z/OS
  - Version 6 with PTF UQ93891
  - Version 7 with PTF UQ93889
  - Version 8 with PTF UQ93890
  - Versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
  - Version 8.2 (equivalent to Version 8.1, Fix Pack 7) and later

This attribute is not supported when accessing Informix database server.

**Note:** This is an IBM defined extension.

### SQL\_ATTR\_STMT\_CONCENTRATOR

Specifies whether dynamic statements that contain literal values use the statement cache.

- **SQL\_ATTR\_STMT\_CONCENTRATOR\_OFF** - The statement concentrator behavior is disabled.
- **SQL\_ATTR\_STMT\_CONCENTRATOR\_WITH\_LITERALS** - The statement concentrator with literal behavior is enabled for situations that are supported by the server. For example, the statement concentrator is not enabled if the statement has parameter markers, named parameter markers, or a mix of literals, parameter markers, and named parameter markers.

Setting the “StmtConcentrator CLI/ODBC configuration keyword” on page 403 is an alternative method of specifying this behavior.

### SQL\_ATTR\_STREAM\_GETDATA

A 32-bit unsigned integer that indicates if the data output stream for the SQLGetData() function will be optimized. The values are:

- **0 (default):** CLI buffers all the data on the client.
- **1:** For applications that do not need to buffer data and are querying data on a server that supports Dynamic Data Format, also known as progressive streaming, specify 1 to indicate that data buffering is not required. The CLI client will optimize the data output stream.

This keyword is ignored if Dynamic Data Format is not supported by the server.

If StreamGetData is set to 1 and CLI cannot determine the number of bytes still available to return in the output buffer, SQLGetData() returns SQL\_NO\_TOTAL (-4) as the length when truncation occurs. Otherwise, SQLGetData() returns the number of bytes still available.

Setting the “StreamGetData CLI/ODBC configuration keyword” on page 403 is an alternative method of specifying this behavior.

### SQL\_ATTR\_SYNC\_POINT

This attribute is deprecated in DB2 UDB Version 8.

**SQL\_ATTR\_TRACE**

This connection attribute can be set by an application for the ODBC Driver Manager. Any attempt to set this connection attribute for the CLI Driver will result in an SQLSTATE of HYC00 (Driver not capable).

Instead of using this connection attribute, the CLI trace facility can be set using the “Trace CLI/ODBC configuration keyword” on page 407. Alternatively, the environment attribute SQL\_ATTR\_TRACE can be used to configure tracing features. Note that the environment attribute does not use the same syntax as the ODBC Driver Manager’s connection attribute.

**SQL\_ATTR\_TRACEFILE**

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

Instead of using this attribute, the CLI trace file name is set using the “TraceFileName CLI/ODBC configuration keyword” on page 413.

**SQL\_ATTR\_TRANSLATE\_LIB**

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute on other platforms will result in an SQLSTATE of HYC00 (Driver not capable).

**SQL\_ATTR\_TRANSLATE\_OPTION**

This connection attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute on other platforms will result in an SQLSTATE of HYC00 (Driver not capable).

**SQL\_ATTR\_TRUSTED\_CONTEXT\_PASSWORD**

A user-defined string containing a password. Use this attribute if the database server requires a password when switching users on a trusted connection. Set this attribute after setting the attribute SQL\_ATTR\_TRUSTED\_CONTEXT\_USERID and before executing any SQL statements that access the database server. If SQL\_ATTR\_TRUSTED\_CONTEXT\_USERID is not set before setting this attribute, an error (CLI0198E) is returned.

This attribute is not supported when accessing Informix database server.

**SQL\_ATTR\_TRUSTED\_CONTEXT\_USERID**

A user-defined string containing a user ID. Use this on existing trusted connections to switch users. Do not use it when creating a trusted connection.

When SQL\_ATTR\_TRUSTED\_CONTEXT\_USERID attribute is set, the user switch will occur next time that you execute an SQL statement that accesses the database server. (SQLSetConnectAttr does not access the database server.) If the user switch is successful the user ID in this attribute becomes the new user of the connection. If the user switch fails the call that initiated the switch will return an error indicating the reason for the failure.

The user ID must be a valid authorization ID on the database server unless you are using an identity server, in which case you can use any user name recognized by the identity server. (If you are using an identity server see also SQL\_ATTR\_USER\_REGISTRY\_NAME.)

If you set this attribute while the connection handle is not yet connected to a database or if the connection is not a trusted connection then an error (CLI0197E) is returned.

## Connection attributes (CLI) list

This attribute is not supported when accessing Informix database server.

### SQL\_ATTR\_TXN\_ISOLATION

A 32-bit bitmask that sets the transaction isolation level for the current connection referenced by *ConnectionHandle*. The valid values for *ValuePtr* can be determined at run time by calling `SQLGetInfo()` with *fnfoType* set to `SQL_TXN_ISOLATION_OPTIONS`. The following values are accepted by CLI, but each server might support only a subset of these isolation levels:

- `SQL_TXN_READ_UNCOMMITTED` - Dirty reads, non-repeatable reads, and phantom reads are possible.
- **`SQL_TXN_READ_COMMITTED` (default)** - Dirty reads are not possible. Non-repeatable reads and phantom reads are possible.
- `SQL_TXN_REPEATABLE_READ` - Dirty reads and reads that cannot be repeated are not possible. Phantoms are possible.
- `SQL_TXN_SERIALIZABLE` - Transactions can be serialized. Dirty reads, non-repeatable reads, and phantoms are not possible.
- `SQL_TXN_NOCOMMIT` - Any changes are effectively committed at the end of a successful operation; no explicit commit or rollback is allowed. This is analogous to autocommit. This is not an SQL92 isolation level, but an IBM defined extension, supported only by DB2 UDB for AS/400.

In IBM terminology,

- `SQL_TXN_READ_UNCOMMITTED` is Uncommitted Read;
- `SQL_TXN_READ_COMMITTED` is Cursor Stability;
- `SQL_TXN_REPEATABLE_READ` is Read Stability;
- `SQL_TXN_SERIALIZABLE` is Repeatable Read.

This option cannot be specified while there is an open cursor on any statement handle, or an outstanding transaction for this connection; otherwise, `SQL_ERROR` is returned on the function call (`SQLSTATE S1011`).

This attribute (or corresponding keyword) is only applicable if the default isolation level is used. If the application specifically set the isolation level then this attribute has no effect.

**Note:** There is an IBM extension that permits the setting of transaction isolation levels on a per statement handle basis. See the `SQL_ATTR_STMTTXN_ISOLATION` statement attribute.

### SQL\_ATTR\_USE\_TRUSTED\_CONTEXT

When connecting to a DB2 database server that supports trusted contexts, set this attribute if you want the connection you are creating to be a trusted connection. If this attribute is set to `SQL_TRUE` and the database server determines that the connection can be trusted then the connection is a trusted connection. If this attribute is not set, if it is set to `SQL_FALSE`, if the database server does not support trusted contexts, or if the database server determines that the connection cannot be trusted then a regular connection is created instead and a warning (`SQLSTATE 01679`) is returned. This value can only be specified before the connection is established either for the first time or following a call to the `SQLDisconnect()` function.

### SQL\_ATTR\_USER\_REGISTRY\_NAME

This attribute is only used when authenticating a user on a server that is using an identity mapping service. It is set to a user-defined string that names an identity mapping registry. The format of the registry name varies

depending on the identity mapping service used. By providing this attribute you tell the server that the user name provided can be found in this registry.

The `SQL_ATTR_USER_REGISTRY_NAME` attribute is used on subsequent attempts to establish a normal connection, establish a trusted connection, or switch the user ID on a trusted connection.

This attribute is not supported when accessing Informix database server.

### **SQL\_ATTR\_WCHARTYPE**

A 32-bit integer that specifies, in a double-byte environment, which `wchar_t` (SQLDBCHAR) character format you want to use in your application. This option provides you the flexibility to choose between having your `wchar_t` data in multi-byte format or in wide-character format. There two possible values for this option:

- **SQL\_WCHARTYPE\_CONVERT**: character codes are converted between the graphic SQL data in the database and the application variable. This allows your application to fully exploit the ANSI C mechanisms for dealing with wide character strings (for example, L-literals, 'wc' string functions) without having to explicitly convert the data to multi-byte format before communicating with the database. The disadvantage is that the implicit conversions might have an impact on the runtime performance of your application, and might increase memory requirements. If you want `WCHARTYPE_CONVERT` behavior then define the C preprocessor macro `SQL_WCHART_CONVERT` at compile time. This ensures that certain definitions in the DB2 header files use the data type `wchar_t` instead of `sqldbchar`.
- **SQL\_WCHARTYPE\_NOCONVERT (default)**: no implicit character code conversion occurs between the application and the database. Data in the application variable is sent to and received from the database as unaltered DBCS characters. This allows the application to have improved performance, but the disadvantage is that the application must either refrain from using wide-character data in `wchar_t` (SQLDBCHAR) application variables, or it must explicitly call the `wcstombs()` and `mbstowcs()` ANSI C functions to convert the data to and from multi-byte format when exchanging data with the database.

**Note:** This is an IBM defined extension.

### **SQL\_ATTR\_XML\_DECLARATION**

A 32-bit unsigned integer that specifies which elements of an XML declaration are added to XML data when it is implicitly serialized. This attribute does not affect the result of the `XMLSERIALIZE` function. Set this attribute to the sum of each component required:

- 0: No declarations or byte order marks (BOMs) are added to the output buffer.
- 1: A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16 or UTF-32. (Although a UTF-8 BOM exists, DB2 does not generate it, even if the target encoding is UTF-8.)
- 2: A minimal XML declaration is generated, containing only the XML version.

## Connection attributes (CLI) list

- 4: An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this attribute.

Attempts to set any other value using `SQLSetConnectAttr()` or `SQLSetConnectOption()` results in a CLI0191E (SQLSTATE HY024) error, and the value remains unchanged.

The default setting is 7, which indicates that a BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.

This setting affects any statement handles allocated after the value is changed. Existing statement handles retain their original values.

This attribute is not supported when accessing Informix database server.

---

## Statement attributes (CLI) list

The currently defined attributes and the version of CLI or ODBC in which they were introduced are shown below; it is expected that more will be defined to take advantage of different data sources.

### SQL\_ATTR\_ALLOW\_INTERLEAVED\_GETDATA

Specifies whether the application can call `SQLGetData()` for previously accessed LOB columns and maintain the data offset position from the previous call to `SQLGetData()` when querying data servers that support Dynamic Data Format. This attribute has one of the following values:

- `SQL_ALLOW_INTERLEAVED_GETDATA_OFF` - This default setting does not allow applications to call `SQLGetData()` for previously accessed LOB columns.
- `SQL_ALLOW_INTERLEAVED_GETDATA_ON` - This keyword only affects connections to database servers that support Dynamic Data Format, also known as progressive streaming. Specify this option to allow applications to call `SQLGetData()` for previously accessed LOB columns and start reading LOB data from where the application stopped reading during the previous read.

Setting the “AllowInterleavedGetData CLI/ODBC configuration keyword” on page 325 is an alternative method of specifying this behavior at the connection level.

`SQL_ATTR_ALLOW_INTERLEAVED_GETDATA` connection attribute is not supported with an IDS data server.

### SQL\_ATTR\_APP\_PARAM\_DESC

The handle to the APD for subsequent calls to `SQLExecute()` and `SQLExecDirect()` on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement was initially allocated. If this attribute is set to `SQL_NULL_DESC`, an explicitly allocated APD handle that was previously associated with the statement handle is dissociated from it, and the statement handle reverts to the implicitly allocated APD handle.

This attribute cannot be set to a descriptor handle that was implicitly allocated for another statement or to another descriptor handle that was implicitly set on the same statement; implicitly allocated descriptor handles cannot be associated with more than one statement or descriptor handle.

This attribute cannot be set at the connection level.

#### SQL\_ATTR\_APP\_ROW\_DESC

The handle to the ARD for subsequent fetches on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement was initially allocated. If this attribute is set to SQL\_NULL\_DESC, an explicitly allocated ARD handle that was previously associated with the statement handle is dissociated from it, and the statement handle reverts to the implicitly allocated ARD handle.

This attribute cannot be set to a descriptor handle that was implicitly allocated for another statement or to another descriptor handle that was implicitly set on the same statement; implicitly allocated descriptor handles cannot be associated with more than one statement or descriptor handle.

This attribute cannot be set at the connection level.

#### SQL\_ATTR\_APP\_USES\_LOB\_LOCATOR

A 32-bit unsigned integer that indicates if applications are using LOB locators. This attribute has either of the following values:

- **1 (default):** Indicates that applications are using LOB locators.
- **0:** For applications that do not use LOB locators and are querying data on a server that supports Dynamic Data Format, also known as progressive streaming, specify 0 to indicate that LOB locators are not used and allow the return of LOB data to be optimized.

This keyword is ignored for stored procedure result sets.

If the keyword is set to 0 and an application binds a LOB locator to a result set that uses SQLBindCol(), an Invalid conversion error is returned by the SQLFetch() function.

Setting the “AppUsesLOBLocator CLI/ODBC configuration keyword” on page 327 is an alternative method of specifying this behavior.

#### SQL\_ATTR\_ASYNC\_ENABLE

A 32-bit integer value that specifies whether a function called with the specified statement is executed asynchronously:

- **SQL\_ATTR\_ASYNC\_ENABLE\_OFF** = Off (the default)
- **SQL\_ATTR\_ASYNC\_ENABLE\_ON** = On

After a function has been called asynchronously, only the original function, SQLAllocHandle(), SQLCancel(), SQLSetStmtAttr(), SQLGetDiagField(), SQLGetDiagRec(), or SQLGetFunctions() can be called on the statement handle, until the original function returns a code other than SQL\_STILL\_EXECUTING. Any other function called on any other statement handle under the same connection returns SQL\_ERROR with an SQLSTATE of HY010 (Function sequence error).

Because CLI supports statement level asynchronous-execution, the statement attribute SQL\_ATTR\_ASYNC\_ENABLE can be set. Its initial value is the same as the value of the connection level attribute with the same name at the time the statement handle was allocated.

The following functions can be executed asynchronously:

SQLBulkOperations(), SQLColAttribute(), SQLColumnPrivileges(), SQLColumns(), SQLDescribeCol(), SQLDescribeParam(), SQLExecDirect(), SQLExecute(), SQLExtendedFetch(), SQLExtendedPrepare(), SQLFetch(), SQLFetchScroll(), SQLForeignKeys(), SQLGetData(), SQLGetLength(), SQLGetPosition(), SQLMoreResults(), SQLNumResultCols(), SQLParamData(),

## Statement attributes (CLI) list

SQLPrepare(), SQLPrimaryKeys(), SQLProcedureColumns(), SQLProcedures(), SQLRowCount(), SQLSetPos(), SQLSpecialColumns(), SQLStatistics(), SQLTablePrivileges(), SQLTables().

**Note:** Any Unicode equivalent of a function stated above can be called asynchronously. Starting from Version 9.7, Fix Pack 4, this attribute can also be used with SQL\_ATTR\_USE\_LOAD\_API.

### SQL\_ATTR\_BLOCK\_FOR\_NROWS

A 32-bit integer that specifies the desired block size, in rows, to be returned by the server when fetching a result set. For large read-only result sets consisting of one or more data blocks, a large block size can improve performance by reducing the number of synchronous server block requests made by the client. The default value is 0 which means the default block size is returned by the server.

### SQL\_ATTR\_BLOCK\_LOBS

A Boolean attribute that specifies if blocking of result sets returning LOB data types is enabled. By default, this attribute is set to 0 (false), however, when set to 1 (true) and when accessing a server that supports blocking of result sets returning LOB data types, all of the LOB data associated with rows that fit completely within a single query block are returned in a single fetch request.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_CALL\_RETURN

A read-only attribute to be retrieved after executing a stored procedure. The value returned from this attribute is -1 if the stored procedure failed to execute (for example, if the library containing the stored procedure executable cannot be found). If the stored procedure executed successfully but has a negative return code (for example, if data truncation occurred when inserting data into a table), then SQL\_ATTR\_CALL\_RETURN returns the value that was set in the sqlerrd (1) field of the SQLCA when the stored procedure was executed.

### SQL\_ATTR\_CHAINING\_BEGIN

A 32-bit integer which specifies that DB2 chains together SQLExecute() requests for a single prepared statement before sending the requests to the server; this feature is referred to as CLI array input chaining. All SQLExecute() requests associated with a prepared statement are not sent to the server until either the SQL\_ATTR\_CHAINING\_END statement attribute is set, or the available buffer space is consumed by rows that have been chained. The size of this buffer is defined by the **aslheapsz** database manager configuration parameter for local client applications, or the **rqrioblk** database manager configuration parameter for client/server configurations.

This attribute can be used with the CLI/ODBC configuration keyword **ArrayInputChain** to effect array input without needing to specify the array size. See the documentation for **ArrayInputChain** for more information.

**Note:** The specific 32-bit integer value that is set with this attribute is not significant to CLI. Setting this attribute to any 32-bit integer value enables the CLI array input chaining feature.

### SQL\_ATTR\_CHAINING\_END

A 32-bit integer which specifies that the CLI array input chaining behavior enabled earlier, with the setting of the SQL\_ATTR\_CHAINING\_BEGIN



statement attribute, ends. Setting `SQL_ATTR_CHAINING_END` causes all chained `SQLExecute()` requests to be sent to the server. After this attribute is set, `SQLRowCount()` can be called to determine the total row count for all `SQLExecute()` statements that were chained between the `SQL_ATTR_CHAINING_BEGIN` and `SQL_ATTR_CHAINING_END` pair. Error diagnostic information for the chained statements becomes available after the `SQL_ATTR_CHAINING_END` attribute is set.

This attribute can be used with the CLI configuration keyword **ArrayInputChain** to affect array input without needing to specify the array size. See the documentation for **ArrayInputChain** for more information.

**Note:** The specific 32-bit integer value that is set with this attribute is not significant to CLI. Setting this attribute to any 32-bit integer value disables the CLI array input chaining feature that was enabled when `SQL_ATTR_CHAINING_BEGIN` was set.

### **SQL\_ATTR\_CLIENT\_LOB\_BUFFERING**

Specifies whether LOB locators or the underlying LOB data is returned in a result set for LOB columns that are not bound. By default, locators are returned. If an application usually fetches unbound LOBs and then must retrieve the underlying LOB data, the application performance can be improved by retrieving the LOB data from the outset. This action reduces the number of synchronous waits and network flows. The possible values for this attribute are:

- `SQL_CLIENTLOB_USE_LOCATORS` (default) - LOB locators are returned
- `SQL_CLIENTLOB_BUFFER_UNBOUND_LOBS` - actual LOB data is returned

### **SQL\_ATTR\_CLOSE\_BEHAVIOR**

A 32-bit integer that specifies whether the DB2 server should attempt to release read locks acquired during a cursor's operation when the cursor is closed. It can be set to either:

- `SQL_CC_NO_RELEASE` - read locks are not released. This is the default.
- `SQL_CC_RELEASE` - read locks are released.

For cursors opened with isolation UR or CS, read locks are not held after a cursor moves off a row. For cursors opened with isolation RS or RR, `SQL_ATTR_CLOSE_BEHAVIOR` modifies some of those isolation levels, and an RR cursor might experience nonrepeatable reads or phantom reads.

If a cursor that is originally RR or RS is reopened after being closed with `SQL_ATTR_CLOSE_BEHAVIOR` then new read locks are acquired.

This attribute can also be set at the connection level, however when set at the connection level, it only affects cursor behavior for statement handles that are opened after this attribute is set.

See the `SQLCloseCursor()` function for more information.

This attribute is not supported when accessing IDS data servers.

### **SQL\_ATTR\_CLOSEOPEN**

To reduce the time it takes to open and close cursors, DB2 automatically closes an open cursor if a second cursor is opened using the same handle. Network flow is therefore reduced when the close request is chained with the open request and the two statements are combined into one network request (instead of two requests).

## Statement attributes (CLI) list

- 0 = DB2 acts as a regular ODBC data source: Do not chain the close and open statements, return an error if there is an open cursor. This behavior is the default.
- 1 = Chain the close and open statements.

Previous CLI applications do not benefit from this default because they are designed to explicitly close the cursor. New applications, however, can take advantage of this behavior by not closing the cursors explicitly, but by allowing CLI to close the cursor on subsequent open requests.

### SQL\_ATTR\_COLUMNWISE\_MRI

A 32-bit unsigned integer that enables CLI applications connected to DB2 for z/OS servers to convert array input chaining into column-wise array input for INSERT operations. This attribute is available starting in Version 9.7 Fix Pack 5. The possible values are as follows:

- **SQL\_COLUMNWISE\_MRI\_OFF (default):** CLI does not convert chaining data to column-wise array input.
- **SQL\_COLUMNWISE\_MRI\_ON:** CLI converts array input chaining to column-wise array input. The Multi-Row Insert (MRI) feature in DB2 for z/OS expects data to be in column-wise array form. If your application uses array input chaining, this conversion helps you optimize your application performance because data is sent in a compact array form each time you call `SQLExecute()`. For more information about array input chaining, see `SQL_ATTR_CHAINING_BEGIN`.

For non-DB2 for z/OS servers, CLI automatically converts chaining data to row-wise array input and setting this attribute has no effect.

The conversion is not performed in the following cases:

- Bind parameters with a LOB data type such as `SQL_CLOB`, `SQL_BLOB`, `SQL_LONGVARBINARY`, `SQL_LONGVARGRAPHIC`, `SQL_DBCLOB`, or `SQL_XML`.
- Bind input data-at-execute parameters by setting their value to `SQL_DATA_AT_EXEC` to pass data to INSERT operations by calling the `SQLPutData()` and `SQLParamData()` functions.
- Space to store all the application data in the internal buffers is not available.

### SQL\_ATTR\_CONCURRENCY

A 32-bit integer value that specifies the cursor concurrency:

- `SQL_CONCUR_READ_ONLY` = Cursor is read-only. No updates are allowed. Supported by forward-only, static and keyset cursors.
- `SQL_CONCUR_LOCK` = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated. Supported by forward-only and keyset cursors.
- `SQL_CONCUR_VALUES` = Cursor uses optimistic concurrency control, comparing values.

The default value for `SQL_ATTR_CONCURRENCY` is `SQL_CONCUR_READ_ONLY` for static and forward-only cursors. The default for a keyset cursor is `SQL_CONCUR_VALUES`.

This attribute cannot be specified for an open cursor.

If the `SQL_ATTR_CURSOR_TYPE` *Attribute* is changed to a type that does not support the current value of `SQL_ATTR_CONCURRENCY`, the value of `SQL_ATTR_CONCURRENCY` is changed at execution time, and a warning issued when `SQLExecuteDirect()` or `SQLPrepare()` is called.

If a `SELECT FOR UPDATE` statement is executed while the value of `SQL_ATTR_CONCURRENCY` is set to `SQL_CONCUR_READ_ONLY`, an error is returned. If the value of `SQL_ATTR_CONCURRENCY` is changed to a value that is supported for some value of `SQL_ATTR_CURSOR_TYPE`, but not for the current value of `SQL_ATTR_CURSOR_TYPE`, the value of `SQL_ATTR_CURSOR_TYPE` is changed at execution time, and `SQLSTATE 01S02` (Option value changed) is issued when `SQLExecDirect()` or `SQLPrepare()` is called.

If the value of `SQL_ATTR_CONCURRENCY` is set to `SQL_CONCUR_LOCK`, this value is promoted to `SQL_CONCUR_VALUES` when all the following conditions are met:

- `SQL_ROWSET_SIZE` OR `SQL_ATTR_ROW_ARRAY_SIZE` is greater than 1.
- The data source is a database on a DB2 Database for Linux, UNIX, and Windows server.
- The **PATCH2** configuration keyword is set to 73.

If the specified concurrency is not supported by the data source, then CLI substitutes a different concurrency and returns `SQLSTATE 01S02` (Option value changed). The order of substitution depends on the cursor type:

- Forward-Only: `SQL_CONCUR_LOCK` is substituted for `SQL_CONCUR_ROWVER` and `SQL_CONCUR_VALUES`
- Static: only `SQL_CONCUR_READ_ONLY` is valid
- Keyset: `SQL_CONCUR_VALUES` is substituted for `SQL_CONCUR_ROWVER`

**Note:** The following value has also been defined by ODBC, but is not supported by CLI

- `SQL_CONCUR_ROWVER` = Cursor uses optimistic concurrency control.

### **SQL\_ATTR\_CURSOR\_HOLD**

A 32-bit integer which specifies whether the cursor associated with this *StatementHandle* is preserved in the same position as before the `COMMIT` operation, and whether the application can fetch without executing the statement again.

- **SQL\_CURSOR\_HOLD\_ON** (this is the default)
- **SQL\_CURSOR\_HOLD\_OFF**

The default value when an *StatementHandle* is first allocated is `SQL_CURSOR_HOLD_ON`.

This option cannot be specified while there is an open cursor on this *StatementHandle*.

The default cursor hold mode can also be set using the **CURSORHOLD** CLI/ODBC configuration keyword.

**Note:** This option is an IBM extension.

### **SQL\_ATTR\_CURSOR\_SCROLLABLE**

A 32-bit integer that specifies the level of support that the application requires. Setting this attribute affects subsequent calls to `SQLExecute()` and `SQLExecDirect()`. The supported values are:

- **SQL\_NONSCROLLABLE** = Scrollable cursors are not required on the statement handle. If the application calls `SQLFetchScroll()` on this handle, the only valid value of `FetchOrientation()` is `SQL_FETCH_NEXT`. This value is the default.

## Statement attributes (CLI) list

- **SQL\_SCROLLABLE** = Scrollable cursors are required on the statement handle. When calling `SQLFetchScroll()`, the application can specify any valid value of *FetchOrientation*, achieving cursor positioning in modes other than the sequential mode.

### **SQL\_ATTR\_CURSOR\_SENSITIVITY**

A 32-bit integer that specifies whether cursors on the statement handle make visible the changes made to a result set by another cursor. Setting this attribute affects subsequent calls to `SQLExecute()` and `SQLExecuteDirect()`. The supported values are:

- **SQL\_UNSPECIFIED** = It is unspecified what the cursor type is and whether cursors on the statement handle make visible the changes made to a result set by another cursor. Cursors on the statement handle might make visible none, some or all such changes. This value is the default.
- **SQL\_INSENSITIVE** = All cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor. Insensitive cursors are read-only. This corresponds to a static cursor which has a concurrency that is read-only.
- **SQL\_SENSITIVE** = All cursors on the statement handle make visible all changes made to a result by another cursor.

### **SQL\_ATTR\_CURSOR\_TYPE**

A 32-bit integer value that specifies the cursor type. The supported values are:

- **SQL\_CURSOR\_FORWARD\_ONLY** = The cursor only scrolls forward. This is the default.
- **SQL\_CURSOR\_STATIC** = The data in the result set is static.
- **SQL\_CURSOR\_KEYSET\_DRIVEN** = CLI supports a pure keyset cursor. The `SQL_KEYSET_SIZE` statement attribute is ignored. To limit the size of the keyset the application must limit the size of the result set by setting the `SQL_ATTR_MAX_ROWS` attribute to a value other than 0.
- **SQL\_CURSOR\_DYNAMIC** = A dynamic scrollable cursor detects all changes (inserts, deletes and updates) to the result set, and make insertions, deletions and updates to the result set. Dynamic cursors are only supported when accessing servers which are DB2 for z/OS Version 8.1 and later.

This option cannot be specified for an open cursor.

If the specified cursor type is not supported by the data source, CLI substitutes a different cursor type and returns `SQLSTATE 01S02` (Option value changed). For a mixed or dynamic cursor, CLI substitutes, in order, a keyset-driven or static cursor.

### **SQL\_ATTR\_DB2\_NOBINDOUT**

A Boolean attribute that specifies when and where the client performs data conversion and related tasks during a fetch operation. The default value of this attribute is 0 (false) and should only be set to 1 (true) when connected to a federated database.

This attribute is not supported when accessing IDS data servers.

### **SQL\_ATTR\_DEFERRED\_PREPARE**

Specifies whether the `PREPARE` request is deferred until the corresponding `execute` request is issued.

- **SQL\_DEFERRED\_PREPARE\_OFF** = Disable deferred prepare. The `PREPARE` request is executed the moment it is issued.

- **SQL\_DEFERRED\_PREPARE\_ON** (default) = Enable deferred prepare. Defer the execution of the PREPARE request until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance.

If the target DB2 database or the DDCS gateway does not support deferred prepare, the client disables deferred prepare for that connection.

**Note:** When deferred prepare is enabled, the row and cost estimates normally returned in the SQLERRD(3) and SQLERRD(4) of the SQLCA of a PREPARE statement might become zeros. This might be of concern to users who want to use these values to decide whether to continue the SQL statement.

The default deferred prepare mode can also be set using the **DEFERREDPREPARE** CLI/ODBC configuration keyword.

**Note:** This is an IBM defined extension.

### **SQL\_ATTR\_EARLYCLOSE**

Specifies whether the temporary cursor on the server can be automatically closed, without closing the cursor on the client, when the last record is sent to the client.

- **SQL\_EARLYCLOSE\_OFF** = Do not close the temporary cursor on the server early.
- **SQL\_EARLYCLOSE\_ON** = Close the temporary cursor on the server early (default).

This saves a network request by not issuing the statement to explicitly close the cursor because it knows that it has already been closed.

Having this option on speeds up applications that use many small result sets.

The EARLYCLOSE feature is not used if the cursor type is anything other than **SQL\_CURSOR\_FORWARD\_ONLY**.

**Note:** This is an IBM defined extension.

### **SQL\_ATTR\_ENABLE\_AUTO\_IPD**

A 32-bit integer value that specifies whether automatic population of the IPD is performed:

- **SQL\_TRUE** = Turns on automatic population of the IPD after a call to **SQLPrepare()**.
- **SQL\_FALSE** = Turns off automatic population of the IPD after a call to **SQLPrepare()**.

The default value of the statement attribute **SQL\_ATTR\_ENABLE\_AUTO\_IPD** is equal to the value of the connection attribute **SQL\_ATTR\_AUTO\_IPD**.

If the connection attribute **SQL\_ATTR\_AUTO\_IPD** is **SQL\_FALSE**, the statement attribute **SQL\_ATTR\_ENABLE\_AUTO\_IPD** cannot be set to **SQL\_TRUE**.

### **SQL\_ATTR\_EXTENDED\_INDICATORS**

A 32-bit integer that eliminates the need to indicate the position in the SQL statement where the contents of the application variables are substituted when the statement is executed. This attribute has the following values:

## Statement attributes (CLI) list

- `SQL_EXTENDED_INDICATOR_ENABLE`: Enables users to specify values to signify `SQL_UNASSIGNED` and `SQL_DEFAULT_PARAM` on the `SQLBindParameter / SQLExtendedBind` methods.
- `SQL_EXTENDED_INDICATOR_NOT_SET` (default): The user gets an `InvalidArgument` value error if the `SQL_UNASSIGNED` and `SQL_DEFAULT_PARAM` are not enabled before an application tries to use them.
- Extended indicators support DB2 for Linux, UNIX, and Windows and for DB2 10 for z/OS data servers starts in DB2 Version 9.7 Fix Pack 2. Extended indicators support DB2 for IBM i 7.1 data servers starts in DB2 Version 9.7 Fix Pack 5.

### `SQL_ATTR_FETCH_BOOKMARK_PTR`

A pointer that points to a binary bookmark value. When `SQLFetchScroll()` is called with *FetchOrientation* equal to `SQL_FETCH_BOOKMARK`, CLI picks up the bookmark value from this field. This field defaults to a null pointer.

### `SQL_ATTR_IMP_PARAM_DESC`

The handle to the IPD. The value of this attribute is the descriptor allocated when the statement was initially allocated. The application cannot set this attribute.

This attribute can be retrieved by a call to `SQLGetStmtAttr()`, but not set by a call to `SQLSetStmtAttr()`.

### `SQL_ATTR_IMP_ROW_DESC`

The handle to the IRD. The value of this attribute is the descriptor allocated when the statement was initially allocated. The application cannot set this attribute.

This attribute can be retrieved by a call to `SQLGetStmtAttr()`, but not set by a call to `SQLSetStmtAttr()`.

### `SQL_ATTR_INFO_PROGRAMID`

A user-defined character string, with a maximum length of 80 bytes, that associates an application with a statement. Once this attribute is set, DB2 UDB for z/OS Version 8 and later associates this identifier with any statements inserted into the dynamic SQL statement cache.

This attribute is only supported for CLI applications accessing DB2 UDB for z/OS Version 8 and later or IBM Informix Dynamic Servers (IDS).

### `SQL_ATTR_INSERT_BUFFERING`

This attribute enables buffering insert optimization of partitioned database environments. The possible values are:  
`SQL_ATTR_INSERT_BUFFERING_OFF` (default),  
`SQL_ATTR_INSERT_BUFFERING_ON`, and  
`SQL_ATTR_INSERT_BUFFERING_IGD` (duplicates are ignored).

This attribute is not supported when accessing IDS data servers.

### `SQL_ATTR_INTERLEAVED_PUTDATA`

This attribute allows inserting LOB data with `SQLParamData` and `SQLPutData` in an interleaving fashion. For example:

```
// Set the attribute
SQLSetStmtAttr(hstmt,
               SQL_ATTR_INTERLEAVED_PUTDATA,
               TRUE,
               0);

//Bind the parameters with DATA_AT_EXEC indicator
blobInd = SQL_DATA_AT_EXEC;
```

```

cliRC = SQLBindParameter (hstmt,          /* statement handle */
                          1,             /* parameter marker index */
                          SQL_PARAM_INPUT, /* it's input parameter */
                          SQL_C_CHAR,    /* CLI variable is CHARACTER*/
                          SQL_CLOB,      /* table column is CLOB*/
                          10,            /* length of CLI variable */
                          0,             /* scale of decimal digits*/
                          &data1,        /* pointer to CLI variable*/
                          10,            /* buffer length */
                          &blobInd);

```

```

cliRC = SQLBindParameter (hstmt,          /* statement handle */
                          2,             /* parameter marker index */
                          SQL_PARAM_INPUT, /* it's input parameter */
                          SQL_C_CHAR,    /* CLI variable is CHARACTER*/
                          SQL_CLOB,      /* table column is CLOB*/
                          10,            /* length of CLI variable */
                          0,             /* scale of decimal digits*/
                          &data2,        /* pointer to CLI variable*/
                          10,            /* buffer length */
                          &blobInd);

```

```

SQLExecute (hstmt);
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data2
SQLPutData (hstmt, data2, strlen(data2));
valuePtr = (SQLPOINTER) 1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data1
SQLPutData (hstmt, data1, strlen(data2));
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data2
SQLPutData (hstmt, data2, strlen(data2));
valuePtr = (SQLPOINTER) 1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data1
SQLPutData (hstmt, data1, strlen(data2));

valuePtr = (SQLPOINTER) 0;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

```

This attribute disables any SQLPutData function streaming that is in effect and causes each of the parameter values to be buffered on the client until the data at the SQL\_DATA\_AT\_EXEC is closed with SQLParamData(0).

### SQL\_ATTR\_INTERLEAVED\_STREAM\_PUTDATA

This attribute allows inserting LOB data with SQLParamData and SQLPutData in an interleaving fashion with function streaming. Streaming writes the LOB data directly to the connection-level buffer, bypassing the internal statement-level buffer, for improved performance.

Applications get a "Function Sequence Error (CLI0125E)" error if a new attribute is set in middle of SQLExecute, SQLParamData or SQLPutData. This error is also returned whenever there is an incorrect sequence of SQLParamData and SQLPutData.

If this attribute is set for a statement handle of a connection, any other statement handle of the same connection gets a "Function Sequence Error (CLI0125E)" if an operation is performed that uses the connection buffer until all the data has been sent to the server for that statement handle with

## Statement attributes (CLI) list

the `SQL_ATTR_INTERLEAVED_STREAM_PUTDATA` attribute enabled. All `SQLParamData` and `SQLPutData` calls must be complete for the statement handle with the `SQL_ATTR_INTERLEAVED_STREAM_PUTDATA` attribute enabled before any other statement handle may perform an operation that uses the connection buffer. For more information on limitations when using streaming, see the “StreamPutData CLI/ODBC configuration keyword” on page 404.

To indicate the end of data for all parameters in a set of interleaved parameters, call `SQLParamData` with a parameter number of (0). Applications should explicitly indicate the end of data for all parameters by calling `SQLParamData` with a parameter number of 0.

The end of data for a single parameter is indicated by calling `SQLParamData` with the equivalent negative parameter number. For example, to indicate the end of the data stream for parameter number 4, the application should specify `SQLParamData(-4)`. Applications should always indicate end of data for a parameter using the negative of the parameter number. If applications indicate the end of data for a parameter which is being streamed, CLI is able to stream data for the next parameter. This may result in better performance.

The following example shows interleaving LOB data, how to mark the end of data for a single parameter using negative parameter numbers, and how to indicate the end of data for all parameters using a parameter number of (0):

```
// Set the SQL_ATTR_INTERLEAVED_STREAM_PUTDATA attribute
SQLSetStmtAttr(hstmt, SQL_ATTR_INTERLEAVED_STREAM_PUTDATA, TRUE, 0);

//Bind the parameters with DATA_AT_EXEC indicator
blobInd = SQL_DATA_AT_EXEC;

//declare the statement handle with parameter marker
index value of 1,
//input parameter SQL_PARAM_INPUT, CLI variable type SQL_C_CHAR,
table column type CLOB,
//length of CLI variable 10, scale of decimal digits 10,
and DATA_AT_EXEC indicator
cliRC = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CLOB,
10, 0, &data1, 10, &blobInd);

//declare the next statement handle with
parameter marker index value of 2
cliRC = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CLOB,
10, 0, &data2, 10, &blobInd);

//declare the next statement handle with
parameter marker index value of 3
cliRC = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CLOB,
10, 0, &data3, 10, &blobInd);

SQLExecute (hstmt);

//make parameter 2 active
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 2
SQLPutData (hstmt, data2, strlen(data2));
```



```

//make parameter 1 active
valuePtr = (SQLPOINTER) 1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

// stream data for parameter 1
SQLPutData (hstmt, data1, strlen(data2));

//make parameter 2 active
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 2
SQLPutData (hstmt, data2, strlen(data2));

//make parameter 3 active
valuePtr = (SQLPOINTER) 3;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 3
SQLPutData (hstmt, data1, strlen(data2));

//end of data for parameter 1
valuePtr = (SQLPOINTER) -1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//make parameter 2 active
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//stream the buffered data for parameter 2
SQLPutData (hstmt, data2, strlen(data2));

//make parameter 3 active valuePtr = (SQLPOINTER) 3;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 3
SQLPutData (hstmt, data1, strlen(data2));

//end of data for parameter 3
valuePtr = (SQLPOINTER) -3;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

// indicate end of data for all parameters.
// CLI streams the buffered data for all parameters
valuePtr = (SQLPOINTER) 0;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

```

**SQL\_ATTR\_KEYSET\_SIZE**

CLI supports a pure keyset cursor, therefore the `SQL_ATTR_KEYSET_SIZE` statement attribute is ignored. To limit the size of the keyset the application must limit the size of the result set by setting the `SQL_ATTR_MAX_ROWS` attribute to a value other than 0.

This attribute is not supported when accessing IDS data servers.

**SQL\_ATTR\_LOAD\_INFO**

A pointer to a structure of type `db2LoadStruct`. The `db2LoadStruct` structure is used to specify all applicable `LOAD` options that should be used during `CLI LOAD`. Note that this pointer and all of its embedded pointers should be valid during every `CLI` function call from the time the `SQL_ATTR_USE_LOAD_API` statement attribute is set to the time it is turned off. For this reason, it is recommended that this pointer and its embedded pointers point to dynamically allocated memory rather than locally declared structures.

This attribute is not supported when accessing IDS data servers.

## Statement attributes (CLI) list

### SQL\_ATTR\_LOAD\_MODIFIED\_BY

A pointer to a char string that specifies the file type modifier option to be used during CLI LOAD.

### SQL\_ATTR\_LOAD\_ROWS\_COMMITTED\_PTR

A pointer to an integer that represents the total number of rows processed. This value equals the number of rows successfully loaded and committed to the database, plus the number of skipped and rejected rows. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_LOAD\_ROWS\_DELETED\_PTR

A pointer to an integer that represents the number of duplicate rows deleted. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_LOAD\_ROWS\_LOADED\_PTR

A pointer to an integer that represents the number of rows loaded into the target table. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_LOAD\_ROWS\_READ\_PTR

A pointer to an integer that represents the number of rows read. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_LOAD\_ROWS\_REJECTED\_PTR

A pointer to an integer that represents the number of rows that could not be loaded. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_LOAD\_ROWS\_SKIPPED\_PTR

A pointer to an integer that represents the number of rows skipped before the CLI LOAD operation began. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_LOB\_CACHE\_SIZE

A 32-bit unsigned integer that specifies maximum cache size (in bytes) for LOBs. By default, LOBs are not cached.

See the “LOBCacheSize CLI/ODBC configuration keyword” on page 366 for further usage information.

### SQL\_ATTR\_MAX\_LENGTH

A 32-bit integer value corresponding to the maximum amount of data that can be retrieved from a single character or binary column.

**Note:** SQL\_ATTR\_MAX\_LENGTH should not be used to truncate data. The *BufferLength* argument of SQLBindCol () or SQLGetData () should be used instead for truncating data.

If data is truncated because the value specified for SQL\_ATTR\_MAX\_LENGTH is less than the amount of data available, a SQLGetData () call or fetch returns SQL\_SUCCESS instead of returning

SQL\_SUCCESS\_WITH\_INFO and SQLSTATE 01004 (Data Truncated). The default value for SQL\_ATTR\_MAX\_LENGTH is 0; 0 means that CLI attempts to return all available data for character or binary type data.

### SQL\_ATTR\_MAX\_LOB\_BLOCK\_SIZE

A 32-bit unsigned integer that indicates the maximum size of LOB or XML data block. Specify a positive integer, up to 2 147 483 647. The default setting of 0 indicates that there is no limit to the data block size for LOB or XML data.

During data retrieval, the server includes all of the information for the current row in its reply to the client even if the maximum block size has been reached.

If both **MaxLOBBlockSize** and the **db2set** registry variable **DB2\_MAX\_LOB\_BLOCK\_SIZE** are specified, the value for **MaxLOBBlockSize** is used.

Setting the "MaxLOBBlockSize CLI/ODBC configuration keyword" on page 376 is an alternative method of specifying this behavior.

### SQL\_ATTR\_MAX\_ROWS

A 32-bit integer value corresponding to the maximum number of rows to return to the application from a query. The default value for SQL\_ATTR\_MAX\_ROWS is 0; 0 means all rows are returned.

### SQL\_ATTR\_METADATA\_ID

This statement attribute is defined by ODBC, but is not supported by CLI. Any attempt to set or get this attribute results in an SQLSTATE of HYC00 (Driver not capable).

This attribute is not supported when accessing IDS data servers.

### SQL\_ATTR\_NOSCAN

A 32-bit integer value that specifies whether CLI scans SQL strings for escape clauses. The two permitted values are:

- **SQL\_NOSCAN\_OFF** - SQL strings are scanned for escape clause sequences. This is the default.
- **SQL\_NOSCAN\_ON** - SQL strings are not scanned for escape clauses. Everything is sent directly to the server for processing.

This application can choose to turn off the scanning if it never uses vendor escape sequences in the SQL strings that it sends. Turning off the scanning eliminates some of the overhead processing associated with scanning.

### SQL\_ATTR\_OPTIMIZE\_FOR\_NROWS

A 32-bit integer value. If it is set to an integer larger than 0, "OPTIMIZE FOR n ROWS" clause is appended to every select statement. If set to 0 (the default) this clause is not appended.

The default value can also be set using the **OPTIMIZEFORNROWS** CLI/ODBC configuration keyword.

### SQL\_ATTR\_OPTIMIZE\_SQLCOLUMNS

This attribute has been deprecated.

### SQL\_ATTR\_PARAM\_BIND\_OFFSET\_PTR

A 32-bit integer \* value that points to an offset added to pointers to change binding of dynamic parameters. If this field is non-null, CLI dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record (SQL\_DESC\_DATA\_PTR,

## Statement attributes (CLI) list

SQL\_DESC\_INDICATOR\_PTR, and SQL\_DESC\_OCTET\_LENGTH\_PTR), and uses the resulting pointer values at execute time. It is set to null by default.

The bind offset is always added directly to the SQL\_DESC\_DATA\_PTR, SQL\_DESC\_INDICATOR\_PTR, and SQL\_DESC\_OCTET\_LENGTH\_PTR fields. If the offset is changed to a different value, the new value is added directly to the value in the descriptor field. The new offset is not added to the field value plus any earlier offsets.

Setting this statement attribute sets the SQL\_DESC\_BIND\_OFFSET\_PTR field in the APD header.

### SQL\_ATTR\_PARAM\_BIND\_TYPE

A 32-bit integer value that indicates the binding orientation to be used for dynamic parameters.

This field is set to **SQL\_PARAM\_BIND\_BY\_COLUMN** (the default) to select column-wise binding.

To select row-wise binding, this field is set to the length of the structure or an instance of a buffer that is bound to a set of dynamic parameters. This length must include space for all of the bound parameters and any padding of the structure or buffer to ensure that when the address of a bound parameter is incremented with the specified length, the result points to the beginning of the same parameter in the next set of parameters. When using the sizeof operator in ANSI C, this behavior is guaranteed.

Setting this statement attribute sets the SQL\_DESC\_BIND\_TYPE field in the APD header.

### SQL\_ATTR\_PARAM\_OPERATION\_PTR

A 16-bit unsigned integer \* value that points to an array of 16-bit unsigned integer values used to specify whether or not a parameter should be ignored during execution of an SQL statement. Each value is set to either **SQL\_PARAM\_PROCEED** (for the parameter to be executed) or **SQL\_PARAM\_IGNORE** (for the parameter to be ignored).

A set of parameters can be ignored during processing by setting the status value in the array pointed to by **SQL\_DESC\_ARRAY\_STATUS\_PTR** in the APD to **SQL\_PARAM\_IGNORE**. A set of parameters is processed if its status value is set to **SQL\_PARAM\_PROCEED**, or if no elements in the array are set.

This statement attribute can be set to a null pointer, in which case CLI does not return parameter status values. This attribute can be set at any time, but the new value is not used until the next time **SQLExecDirect()** or **SQLExecute()** is called.

Setting this statement attribute sets the **SQL\_DESC\_ARRAY\_STATUS\_PTR** field in the APD.

### SQL\_ATTR\_PARAM\_STATUS\_PTR

A 16-bit unsigned integer \* value that points to an array of UWORD values containing status information for each row of parameter values after a call to **SQLExecDirect()** or **SQLExecute()**. This field is used only if **SQL\_ATTR\_PARAMSET\_SIZE** is greater than 1.

The status values can contain the following values:

- **SQL\_PARAM\_SUCCESS**: The SQL statement was successfully executed for this set of parameters.

- **SQL\_PARAM\_SUCCESS\_WITH\_INFO:** The SQL statement was successfully executed for this set of parameters; however, warning information is available in the diagnostics data structure.
- **SQL\_PARAM\_ERROR:** There was an error in processing this set of parameters. Additional error information is available in the diagnostics data structure.
- **SQL\_PARAM\_UNUSED:** This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing.
- **SQL\_PARAM\_DIAG\_UNAVAILABLE:** CLI treats arrays of parameters as a monolithic unit and so does not generate this level of error information.

This statement attribute can be set to a null pointer, in which case CLI does not return parameter status values. This attribute can be set at any time, but the new value is not used until the next time `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()` is called.

Setting this statement attribute sets the `SQL_DESC_ARRAY_STATUS_PTR` field in the IPD header.

### **SQL\_ATTR\_PARAMOPT\_ATOMIC**

This is a 32-bit integer value which determines, when `SQLParamOptions()` has been used to specify multiple values for parameter markers, whether the underlying processing should be done via `ATOMIC` or `NOT-ATOMIC` Compound SQL. The possible values are:

- **SQL\_ATOMIC\_YES** - The underlying processing makes use of `ATOMIC` Compound SQL. This is the default if the target database supports `ATOMIC` compound SQL.
- **SQL\_ATOMIC\_NO** - The underlying processing makes use of `NON-ATOMIC` Compound SQL.

Specifying `SQL_ATOMIC_YES` when connected to a server that does not support `ATOMIC` compound SQL results in an error (`SQLSTATE` is `S1C00`).

Specifying `SQL_ATOMIC_YES` when `SQL_PARC_BATCH` is set to `SQL_PARC_BATCH_ENABLE` returns the `CLI0150E` error message. If you want to set `SQL_PARC_BATCH` to `SQL_PARC_BATCH_ENABLE`, you must specify `SQL_ATOMIC_NO`.

### **SQL\_ATTR\_PARAMS\_PROCESSED\_PTR**

A 32-bit unsigned integer \* record field that points to a buffer in which to return the current row number. As each row of parameters is processed, this is set to the number of that row. No row number is returned if this is a null pointer.

Setting this statement attribute sets the `SQL_DESC_ROWS_PROCESSED_PTR` field in the IPD header.

If the call to `SQLExecDirect()` or `SQLExecute()` that fills in the buffer pointed to by this attribute does not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the buffer are undefined.

### **SQL\_ATTR\_PARAMSET\_SIZE**

A 32-bit unsigned integer value that specifies the number of values for each parameter. If `SQL_ATTR_PARAMSET_SIZE` is greater than 1, `SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR` of the APD point to arrays. The cardinality of each array is equal to the value of this field.

## Statement attributes (CLI) list

Setting this statement attribute sets the `SQL_DESC_ARRAY_SIZE` field in the APD header.

Starting in DB2 Version 9.7 Fix Pack 6, array input using `SQL_ATTR_PARAMSET_SIZE`, inside a trusted procedure body, is supported.

### **SQL\_ATTR\_PREFETCH**

This attribute has been deprecated.

### **SQL\_ATTR\_QUERY\_OPTIMIZATION\_LEVEL**

A 32-bit integer value that sets the query optimization level to be used on the next call to `SQLPrepare()`, `SQLExtendedPrepare()`, or `SQLExecDirect()`.

Supported values to use are: -1 (default), 0, 1, 2, 3, 5, 7, and 9.

The `SQL_ATTR_QUERY_OPTIMIZATION_LEVEL` statement attribute does not set the optimization level for IDS data servers. Informix optimizer directives should be used instead. For more information, see *Optimizer directives*

### **SQL\_ATTR\_QUERY\_TIMEOUT**

A 32-bit integer value that is the number of seconds to wait for an SQL statement or XQuery expression to execute before aborting the execution and returning to the application. This option can be set and used to terminate long running queries. The default value of 0 means CLI waits indefinitely for the server to complete execution of the SQL statement. CLI supports non-zero values for all platforms that support multithreading.

When using this attribute against a server which does not have native interrupt support (such as DB2 for z/OS and OS/390, Version 7 and earlier, and DB2 for i), the `INTERRUPT_ENABLED` option must be set when cataloging the DCS database entry for the server.

When the `INTERRUPT_ENABLED` option is set and this attribute is set to a non-zero value, the DB2 for i server drops the connection and rolls back the unit of work. The application receives an SQL30081N error indicating that the connection to the server has been terminated. In order for the application to process additional database requests, the application must establish a new connection with the database server.

The `SQL_ATTR_QUERY_TIMEOUT` can also interrupt a `LOAD`, which returns SQL3005N instead of SQL0952N.

### **SQL\_ATTR\_REOPT**

A 32-bit integer value that enables query optimization for SQL statements that contain special registers or parameter markers. Optimization occurs by using the values available at query execution time for special registers or parameter markers, instead of the default estimates that are chosen by the compiler. The valid values of the attribute are:

- 2 = `SQL_REOPT_NONE`. This is the default. No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for the special registers or parameter markers. The default NULLID package set is used to execute dynamic SQL statements.
- 3 = `SQL_REOPT_ONCE`. Query optimization occurs once at query execution time, when the query is executed for the first time. The NULLIDR1 package set, which is bound with the `REOPT ONCE` bind option, is used.

- 4 = `SQL_REOPT_ALWAYS`. Query optimization or reoptimization occurs at query execution time every time the query is executed. The `NULLIDRA` package set, which is bound with the `REOPT ALWAYS` bind option, is used.

The `NULLIDR1` and `NULLIDRA` are reserved package set names, and when used, `REOPT ONCE` and `REOPT ALWAYS` are implied respectively. These package sets have to be explicitly created with these commands:

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

`SQL_ATTR_REOPT` and `SQL_ATTR_CURRENT_PACKAGE_SET` are mutually exclusive, therefore, if one is set, the other is not allowed.

This attribute is not supported when accessing IDS data servers.

#### **SQL\_ATTR\_RETRIEVE\_DATA**

A 32-bit integer value:

- `SQL_RD_ON` = `SQLFetchScroll()` and in DB2 CLI/v5 and later, `SQLFetch()`, retrieve data after it positions the cursor to the specified location. This is the default.
- `SQL_RD_OFF` = `SQLFetchScroll()` and in DB2 CLI/v5 and later, `SQLFetch()`, do not retrieve data after it positions the cursor.

By setting `SQL_RETRIEVE_DATA` to `SQL_RD_OFF`, an application can verify if a row exists or retrieve a bookmark for the row without incurring the overhead of retrieving rows.

#### **SQL\_ATTR\_RETURN\_USER\_DEFINED\_TYPES**

A Boolean attribute that specifies whether user-defined type columns are reported as the user-defined type or the underlying base type when queried by functions such as `SQLDescribeCol()`. The default value is 0 (false), where columns are reported as the underlying base type.

This attribute is not supported when accessing IDS data servers.

#### **SQL\_ATTR\_ROW\_ARRAY\_SIZE**

A 32-bit integer value that specifies the number of rows in the rowset. This is the number of rows returned by each call to `SQLFetch()` or `SQLFetchScroll()`. The default value is 1.

If the specified rowset size exceeds the maximum rowset size supported by the data source, CLI substitutes that value and returns `SQLSTATE 01S02` (Option value changed).

This option can be specified for an open cursor.

Setting this statement attribute sets the `SQL_DESC_ARRAY_SIZE` field in the ARD header.

#### **SQL\_ATTR\_ROW\_BIND\_OFFSET\_PTR**

A 32-bit integer \* value that points to an offset added to pointers to change binding of column data. If this field is non-null, CLI dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record (`SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR`), and uses the new pointer values when binding. It is set to null by default.

Setting this statement attribute sets the `SQL_DESC_BIND_OFFSET_PTR` field in the ARD header.

## Statement attributes (CLI) list

### SQL\_ATTR\_ROW\_BIND\_TYPE

A 32-bit integer value that sets the binding orientation to be used when `SQLFetch()` or `SQLFetchScroll()` is called on the associated statement. Column-wise binding is selected by supplying the defined constant `SQL_BIND_BY_COLUMN` in *\*ValuePtr*. Row-wise binding is selected by supplying a value in *\*ValuePtr* specifying the length of a structure or an instance of a buffer into which result columns are bound.

The length specified in *\*ValuePtr* must include space for all of the bound columns and any padding of the structure or buffer to ensure that when the address of a bound column is incremented with the specified length, the result points to the beginning of the same column in the next row. When using the `sizeof` operator with structures or unions in ANSI C, this behavior is guaranteed.

Column-wise binding is the default binding orientation for `SQLFetch()` and `SQLFetchScroll()`.

Setting this statement attribute sets the `SQL_DESC_BIND_TYPE` field in the ARD header.

### SQL\_ATTR\_ROW\_NUMBER

A 32-bit integer value that is the number of the current row in the entire result set. If the number of the current row cannot be determined or there is no current row, CLI returns 0.

This attribute can be retrieved by a call to `SQLGetStmtAttr()`, but not set by a call to `SQLSetStmtAttr()`.

### SQL\_ATTR\_ROW\_OPERATION\_PTR

A 16-bit unsigned integer \* value that points to an array of `UDWORD` values used to ignore a row during a bulk operation using `SQLSetPos()`. Each value is set to either `SQL_ROW_PROCEED` (for the row to be included in the bulk operation) or `SQL_ROW_IGNORE` (for the row to be excluded from the bulk operation).

This statement attribute can be set to a null pointer, in which case CLI does not return row status values. This attribute can be set at any time, but the new value is not used until the next time `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()` is called.

Setting this statement attribute sets the `SQL_DESC_ARRAY_STATUS_PTR` field in the ARD.

### SQL\_ATTR\_ROW\_STATUS\_PTR

A 16-bit unsigned integer \* value that points to an array of `UWORD` values containing row status values after a call to `SQLFetch()` or `SQLFetchScroll()`. The array has as many elements as there are rows in the rowset.

This statement attribute can be set to a null pointer, in which case CLI does not return row status values. This attribute can be set at any time, but the new value is not used until the next time `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()` is called.

Setting this statement attribute sets the `SQL_DESC_ARRAY_STATUS_PTR` field in the IRD header.

### SQL\_ATTR\_ROWS\_FETCHED\_PTR

A 32-bit unsigned integer \* value that points to a buffer in which to return the number of rows fetched after a call to `SQLFetch()` or `SQLFetchScroll()`.



Setting this statement attribute sets the `SQL_DESC_ROWS_PROCESSED_PTR` field in the IRD header.

This attribute is mapped by CLI to the *RowCountPtr* array in a call to `SQLExtendedFetch()`.

#### SQL\_ATTR\_ROWCOUNT\_PREFETCH

This attribute enables CLI to determine the number of rows so that the entire result set can be prefetched. This attribute has one of the following values:

- **0 (default):** Off
- **1:** On

If you set `SQL_ATTR_ROWCOUNT_PREFETCH` to 0 and call `SQLRowCount()` using a non-scrollable SELECT-only cursor, the function sets the contents of *RowCountPtr* to -1 because the number of rows is not available until all of the data has been fetched.

If you set `SQL_ATTR_ROWCOUNT_PREFETCH` to 1 and call `SQLRowCount()` using a non-scrollable SELECT-only cursor, the following occurs:

- If you use `SELECT * FROM INSERT | UPDATE | DELETE` statements with forward-only cursors, the row count comes from the SELECT statements. This is different than the rows-affected count that is provided with these cursors without this attribute set.
- All cursor data is prefetched. This might take several round trips to the server and a considerable amount of memory on the client.
- The prefetched data is not discarded; instead, it is used to satisfy the fetch requests by the application.

This attribute is not applicable to scrollable cursors because they can provide a row count.

Specify this attribute before preparing a statement.

**Restriction:** `SQL_ATTR_ROWCOUNT_PREFETCH` is not supported when the cursor contains LOBs or XML.

#### SQL\_ROWSET\_SIZE

CLI applications should now use `SQLFetchScroll()` rather than `SQLExtendedFetch()`. Applications should also use the statement attribute `SQL_ATTR_ROW_ARRAY_SIZE` to set the number of rows in the rowset.

A 32-bit integer value that specifies the number of rows in the rowset. A rowset is the array of rows returned by each call to `SQLExtendedFetch()`. The default value is 1, which is equivalent to making a single `SQLFetch()` call. This option can be specified even when the cursor is open and becomes effective on the next `SQLExtendedFetch()` call.

#### SQL\_ATTR\_SIMULATE\_CURSOR (ODBC 2.0)

This statement attribute is not supported by CLI but is defined by ODBC.

This attribute is not supported when accessing IDS data servers.

#### SQL\_ATTR\_STMT\_CONCENTRATOR

Starting in Version 9.7 Fix Pack 3a, DB2 supports this statement attribute, which specifies whether dynamic statements that contain literal values use the statement cache.

- `SQL_STMT_CONCENTRATOR_OFF` - The statement concentrator behavior is disabled.

## Statement attributes (CLI) list

- **SQL\_STMT\_CONCENTRATOR\_WITH\_LITERALS** - The statement concentrator with literal behavior is enabled for situations that are supported by the server. For example, the statement concentrator is not enabled if the statement has parameter markers, named parameter markers, or a mix of literals, parameter markers, and named parameter markers.

Setting the “*StmtConcentrator CLI/ODBC configuration keyword*” on page 403 is an alternative method of specifying this behavior.

**Note:** When this attribute is used against DB2® for z/OS® servers older than Version 10, the request is ignored.

### **SQL\_ATTR\_STMTTXN\_ISOLATION**

See **SQL\_ATTR\_TXN\_ISOLATION**.

### **SQL\_ATTR\_STREAM\_GETDATA**

A 32-bit unsigned integer that indicates if the data output stream for the `SQLGetData()` function is optimized. The values are:

- **0 (default):** CLI buffers all the data on the client.
- **1:** For applications that do not need to buffer data and are querying data on a server that supports Dynamic Data Format, specify 1 to indicate that data buffering is not required. The CLI client optimizes the data output stream.

This keyword is ignored if Dynamic Data Format is not supported by the server.

If **StreamGetData** is set to 1 and CLI cannot determine the number of bytes still available to return in the output buffer, `SQLGetData()` returns `SQL_NO_TOTAL` (-4) as the length when truncation occurs. Otherwise, `SQLGetData()` returns the number of bytes still available.

Setting the “*StreamGetData CLI/ODBC configuration keyword*” on page 403 is an alternative method of specifying this behavior.

### **SQL\_ATTR\_TXN\_ISOLATION**

A 32-bit integer value that sets the transaction isolation level for the current *StatementHandle*.

This option cannot be set if there is an open cursor on this statement handle (SQLSTATE 24000).

The value `SQL_ATTR_STMTTXN_ISOLATION` is synonymous with `SQL_ATTR_TXN_ISOLATION`. However, since the ODBC Driver Manager rejects the setting of `SQL_ATTR_TXN_ISOLATION` as a statement option, ODBC applications that need to set transaction isolation level on a per statement basis must use the manifest constant `SQL_ATTR_STMTTXN_ISOLATION` instead on the `SQLSetStmtAttr()` call.

The default transaction isolation level can also be set using the **TXNISOLATION** CLI/ODBC configuration keyword.

This attribute (or corresponding keyword) is only applicable if the default isolation level is used for the statement handle. If the application has specifically set the isolation level for the statement handle, then this attribute does not have effect.

**Note:** It is an IBM extension to allow setting this option at the statement level.

**SQL\_ATTR\_USE\_BOOKMARKS**

A 32-bit integer value that specifies whether an application CLI0150E is returned use bookmarks with a cursor:

- **SQL\_UB\_OFF** = Off (the default)
- **SQL\_UB\_VARIABLE** = An application uses bookmarks with a cursor, and CLI provides variable-length bookmarks if they are supported.

To use bookmarks with a cursor, the application must specify this option with the **SQL\_UB\_VARIABLE** value before opening the cursor.

**SQL\_ATTR\_USE\_LOAD\_API**

A 32-bit integer that indicates if the LOAD utility replaces the regular CLI array insert for inserting data. The possible values are:

**SQL\_USE\_LOAD\_OFF**

(Default) Use regular CLI array insert to insert data.

**SQL\_USE\_LOAD\_INSERT**

Use the LOAD utility to append to existing data in the table.

**SQL\_USE\_LOAD\_REPLACE**

Use the LOAD utility to replace existing data in the table.

**SQL\_USE\_LOAD\_RESTART**

Resume a previously failed CLI LOAD operation. If the previous CLI LOAD operation failed while rows were being inserted (that is, before the **SQL\_ATTR\_USE\_LOAD\_API** statement attribute was set to **SQL\_USE\_LOAD\_OFF**), the CLI LOAD feature remains active, and subsequent rows are inserted by the CLI LOAD utility. Otherwise, if the operation failed while CLI LOAD was being turned off, regular CLI array inserts resume after the restarted load completes.

**SQL\_USE\_LOAD\_TERMINATE**

Clean up and undo a previously failed CLI LOAD operation. After setting the statement attribute to this value, regular CLI array inserts resume.

This attribute is not supported when accessing IDS data servers.

**Note:** Starting from Version 9.7, Fix Pack 4, this attribute can be used with **SQL\_ATTR\_ASYNC\_ENABLE**.

**SQL\_ATTR\_XML\_DECLARATION**

A 32-bit unsigned integer that specifies which elements of an XML declaration are added to XML data when it is implicitly serialized. This attribute does not affect the result of the XMLSERIALIZE function.

This attribute can only be specified on a statement handle that has no open cursors associated with it. Attempting to update the value of this attribute while there are open cursors on the statement handle results in a CLI0126E (SQLSTATE HY011) error, and the value remains unchanged.

Set this attribute to the sum of each component required:

- 0** No declarations or byte order marks (BOMs) are added to the output buffer.
- 1** A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16 or UTF-32. (Although a UTF-8 BOM exists, DB2 does not generate it, even if the target encoding is UTF-8.)

## Statement attributes (CLI) list

- 2 A minimal XML declaration is generated, containing only the XML version.
- 4 An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this attribute.

Attempts to set any other value using `SQLSetStmtAttr()` or `SQLSetStmtOption()` results in a CLI0191E (SQLSTATE HY024) error, and the value remains unchanged.

The default setting is 7, which indicates that a BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.

This attribute can also be specified on a connection handle and affects any statement handles allocated after the value is changed. Existing statement handles retain their original values.

This attribute is not supported when accessing IDS data servers.

### **SQL\_ATTR\_XQUERY\_STATEMENT**

A 32-bit integer value that specifies whether the statement associated with the current statement handle is an XQuery expression or an SQL statement or query. This can be used by CLI applications that do not want to prefix an XQuery expression with the "XQUERY" keyword. The supported values are:

#### **SQL\_TRUE**

The next statement executed on the current statement handle is processed as an XQuery expression. If the server does not support XQuery, setting this attribute to `SQL_TRUE` results in a warning, CLI0005W (SQLSTATE 01S02), and the attribute's value is unchanged.

#### **SQL\_FALSE (default)**

The next statement executed on the current statement handle is processed as an SQL statement.

This attribute takes effect on the next `SQLPrepare()` or `SQLExecDirect()` function call.

This attribute is not supported when accessing IDS data servers.

---

## Chapter 5. Descriptor values

---

### Descriptor FieldIdentifier argument values (CLI)

The *FieldIdentifier* argument indicates the descriptor field to be set. A descriptor contains the descriptor header, consisting of the header fields described in the next section, and zero or more descriptor records, consisting of the record fields described in the following section.

#### Header fields

Each descriptor has a header consisting of the following fields.

**SQL\_DESC\_ALLOC\_TYPE [All]** This read-only SQLSMALLINT header field specifies whether the descriptor was allocated automatically by CLI or explicitly by the application. The application can obtain, but not modify, this field. The field is set to SQL\_DESC\_ALLOC\_AUTO if the descriptor was automatically allocated. It is set to SQL\_DESC\_ALLOC\_USER if the descriptor was explicitly allocated by the application.

**SQL\_DESC\_ARRAY\_SIZE [Application descriptors]** In ARDs, this SQLINTEGER header field specifies the number of rows in the rowset. This is the number of rows to be returned by a call to SQLFetch(), SQLFetchScroll(), or SQLSetPos(). The default value is 1. This field can also be set by calling SQLSetStmtAttr() with the SQL\_ATTR\_ROW\_ARRAY\_SIZE attribute.

In APDs, this SQLINTEGER header field specifies the number of values for each parameter.

The default value of this field is 1. If SQL\_DESC\_ARRAY\_SIZE is greater than 1, SQL\_DESC\_DATA\_PTR, SQL\_DESC\_INDICATOR\_PTR, and SQL\_DESC\_OCTET\_LENGTH\_PTR of the APD or ARD point to arrays. The cardinality of each array is equal to the value of this field.

This field in the ARD can also be set by calling SQLSetStmtAttr() with the SQL\_ROWSET\_SIZE attribute. This field in the APD can also be set by calling SQLSetStmtAttr() with the SQL\_ATTR\_PARAMSET\_SIZE attribute.

**SQL\_DESC\_ARRAY\_STATUS\_PTR [All]** For each descriptor type, this SQLUSMALLINT \* header field points to an array of SQLUSMALLINT values. This array is referred to as:

- row status array (IRD)
- parameter status array (IPD)
- row operation array (ARD)
- parameter operation array (APD)

In the IRD, this header field points to a row status array containing status values after a call to SQLFetch(), SQLFetchScroll(), or SQLSetPos(). The array has as many elements as there are rows in the rowset. The application must allocate an array of SQLUSMALLINTs and set this field to point to the array. The field is set to a null pointer by default. CLI will populate the array, unless the SQL\_DESC\_ARRAY\_STATUS\_PTR field is set to a null pointer, in which case no status values are generated and the array is not populated.

## Descriptor FieldIdentifier argument values (CLI)

**Note:** Behavior is undefined if the application sets the elements of the row status array pointed to by the `SQL_DESC_ARRAY_STATUS_PTR` field of the IRD. The array is initially populated by a call to `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()`. If the call did not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the array pointed to by this field are undefined.

The elements in the array can contain the following values:

- `SQL_ROW_SUCCESS`: The row was successfully fetched and has not changed since it was last fetched.
- `SQL_ROW_SUCCESS_WITH_INFO`: The row was successfully fetched and has not changed since it was last fetched. However, a warning was returned about the row.
- `SQL_ROW_ERROR`: An error occurred while fetching the row.
- `SQL_ROW_UPDATED`: The row was successfully fetched and has been updated since it was last fetched. If the row is fetched again, its status is `SQL_ROW_SUCCESS`.
- `SQL_ROW_DELETED`: The row has been deleted since it was last fetched.
- `SQL_ROW_ADDED`: The row was inserted by `SQLSetPos()`. If the row is fetched again, its status is `SQL_ROW_SUCCESS`.
- `SQL_ROW_NOROW`: The rowset overlapped the end of the result set and no row was returned that corresponded to this element of the row status array.

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the `SQL_ATTR_ROW_STATUS_PTR` attribute.

In the IPD, this header field points to a parameter status array containing status information for each set of parameter values after a call to `SQLExecute()` or `SQLExecDirect()`. If the call to `SQLExecute()` or `SQLExecDirect()` did not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the array pointed to by this field are undefined. The application must allocate an array of `SQLUSMALLINT`s and set this field to point to the array. The driver will populate the array, unless the `SQL_DESC_ARRAY_STATUS_PTR` field is set to a null pointer, in which case no status values are generated and the array is not populated.

The elements in the array can contain the following values:

- `SQL_PARAM_SUCCESS`: The SQL statement was successfully executed for this set of parameters.
- `SQL_PARAM_SUCCESS_WITH_INFO`: The SQL statement was successfully executed for this set of parameters; however, warning information is available in the diagnostics data structure.
- `SQL_PARAM_ERROR`: An error occurred in processing this set of parameters. Additional error information is available in the diagnostics data structure.
- `SQL_PARAM_UNUSED`: This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing.
- `SQL_PARAM_DIAG_UNAVAILABLE`: Diagnostic information is not available. An example of this is when CLI treats arrays of parameters as a monolithic unit and so does not generate this level of error information.

This field in the APD can also be set by calling `SQLSetStmtAttr()` with the `SQL_ATTR_PARAM_STATUS_PTR` attribute.

## Descriptor FieldIdentifier argument values (CLI)

In the ARD, this header field points to a row operation array of values that can be set by the application to indicate whether this row is to be ignored for `SQLSetPos()` operations.

The elements in the array can contain the following values:

- `SQL_ROW_PROCEED`: The row is included in the bulk operation using `SQLSetPos()`. (This setting does not guarantee that the operation will occur on the row. If the row has the status `SQL_ROW_ERROR` in the IRD row status array, CLI may not be able to perform the operation in the row.)
- `SQL_ROW_IGNORE`: The row is excluded from the bulk operation using `SQLSetPos()`.

If no elements of the array are set, all rows are included in the bulk operation. If the value in the `SQL_DESC_ARRAY_STATUS_PTR` field of the ARD is a null pointer, all rows are included in the bulk operation; the interpretation is the same as if the pointer pointed to a valid array and all elements of the array were `SQL_ROW_PROCEED`. If an element in the array is set to `SQL_ROW_IGNORE`, the value in the row status array for the ignored row is not changed.

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the `SQL_ATTR_ROW_OPERATION_PTR` attribute.

In the APD, this header field points to a parameter operation array of values that can be set by the application to indicate whether this set of parameters is to be ignored when `SQLExecute()` or `SQLExecDirect()` is called. The elements in the array can contain the following values:

- `SQL_PARAM_PROCEED`: The set of parameters is included in the `SQLExecute()` or `SQLExecDirect()` call.
- `SQL_PARAM_IGNORE`: The set of parameters is excluded from the `SQLExecute()` or `SQLExecDirect()` call.

If no elements of the array are set, all sets of parameters in the array are used in the `SQLExecute()` or `SQLExecDirect()` calls. If the value in the `SQL_DESC_ARRAY_STATUS_PTR` field of the APD is a null pointer, all sets of parameters are used; the interpretation is the same as if the pointer pointed to a valid array and all elements of the array were `SQL_PARAM_PROCEED`.

This field in the APD can also be set by calling `SQLSetStmtAttr()` with the `SQL_ATTR_PARAM_OPERATION_PTR` attribute.

**SQL\_DESC\_BIND\_OFFSET\_PTR [Application descriptors]** This `SQLINTEGER *` header field points to the bind offset. It is set to a null pointer by default. If this field is not a null pointer, CLI dereferences the pointer and adds the dereferenced value to each of the deferred fields that has a non-null value in the descriptor record (`SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR`) at fetch time, and uses the new pointer values when binding.

The bind offset is always added directly to the values in the `SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR` fields. If the offset is changed to a different value, the new value is still added directly to the value in each descriptor field. The new offset is not added to the field value plus any earlier offset.

## Descriptor FieldIdentifier argument values (CLI)

This field is a *deferred field*: it is not used at the time it is set, but is used at a later time by CLI to retrieve data.

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the `SQL_ATTR_ROW_BIND_OFFSET_PTR` attribute. This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the `SQL_ATTR_PARAM_BIND_OFFSET_PTR` attribute.

**SQL\_DESC\_BIND\_TYPE [Application descriptors]** This SQLINTEGER header field sets the binding orientation to be used for either binding columns or parameters.

In ARDs, this field specifies the binding orientation when `SQLFetchScroll()` is called on the associated statement handle.

To select column-wise binding for columns, this field is set to `SQL_BIND_BY_COLUMN` (the default).

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with `SQL_ATTR_ROW_BIND_TYPE` Attribute.

In APDs, this field specifies the binding orientation to be used for dynamic parameters.

To select column-wise binding for parameters, this field is set to `SQL_BIND_BY_COLUMN` (the default).

This field in the APD can also be set by calling `SQLSetStmtAttr()` with `SQL_ATTR_PARAM_BIND_TYPE` Attribute.

**SQL\_DESC\_COUNT [All]** This SQLSMALLINT header field specifies the one-based index of the highest-numbered record that contains data. When CLI sets the data structure for the descriptor, it must also set the `COUNT` field to show how many records are significant. When an application allocates an instance of this data structure, it does not have to specify how many records to reserve room for. As the application specifies the contents of the records, CLI takes any required action to ensure that the descriptor handle refers to a data structure of adequate size.

`SQL_DESC_COUNT` is not a count of all data columns that are bound (if the field is in an ARD), or all parameters that are bound (in an APD), but the number of the highest-numbered record. If a column or a parameter with a number that is less than the number of the highest-numbered column is unbound (by calling `SQLBindCol()` with the *Target ValuePtr* argument set to a null pointer, or `SQLBindParameter()` with the *Parameter ValuePtr* argument set to a null pointer), `SQL_DESC_COUNT` is not changed. If additional columns or parameters are bound with numbers greater than the highest-numbered record that contains data, CLI automatically increases the value in the `SQL_DESC_COUNT` field. If all columns or parameters are unbound by calling `SQLFreeStmt()` with the `SQL_UNBIND` option, `SQL_DESC_COUNT` is set to 0.

The value in `SQL_DESC_COUNT` can be set explicitly by an application by calling `SQLSetDescField()`. If the value in `SQL_DESC_COUNT` is explicitly decreased, all records with numbers greater than the new value in `SQL_DESC_COUNT` are removed, unbinding the columns. If the value in `SQL_DESC_COUNT` is explicitly set to 0, and the field is in an APD, all parameters are unbound. If the value in



## Descriptor FieldIdentifier argument values (CLI)

SQL\_DESC\_COUNT is explicitly set to 0, and the field is in an ARD, all data buffers except a bound bookmark column are released.

The record count in this field of an ARD does not include a bound bookmark column.

**SQL\_DESC\_ROWS\_PROCESSED\_PTR [Implementation descriptors]** In an IRD, this SQLUIINTEGER \* header field points to a buffer containing the number of rows fetched after a call to SQLFetch() or SQLFetchScroll(), or the number of rows affected in a bulk operation performed by a call to SQLSetPos().

In an IPD, this SQLUIINTEGER \* header field points to a buffer containing the number of the row as each row of parameters is processed. No row number will be returned if this is a null pointer.

SQL\_DESC\_ROWS\_PROCESSED\_PTR is valid only after SQL\_SUCCESS or SQL\_SUCCESS\_WITH\_INFO has been returned after a call to SQLFetch() or SQLFetchScroll() (for an IRD field) or SQLExecute() or SQLExecDirect() (for an IPD field). If the return code is not one of the above, the location pointed to by SQL\_DESC\_ROWS\_PROCESSED\_PTR is undefined. If the call that fills in the buffer pointed to by this field did not return SQL\_SUCCESS or SQL\_SUCCESS\_WITH\_INFO, the contents of the buffer are undefined, unless it returns SQL\_NO\_DATA, in which case the value in the buffer is set to 0.

This field in the ARD can also be set by calling SQLSetStmtAttr() with the SQL\_ATTR\_ROWS\_FETCHED\_PTR attribute. This field in the ARD can also be set by calling SQLSetStmtAttr() with the SQL\_ATTR\_PARAMS\_PROCESSED\_PTR attribute.

The buffer pointed to by this field is allocated by the application. It is a deferred output buffer that is set by CLI. It is set to a null pointer by default.

### Record fields

Each descriptor contains one or more records consisting of fields that define either column data or dynamic parameters, depending on the type of descriptor. Each record is a complete definition of a single column or parameter.

**SQL\_DESC\_AUTO\_UNIQUE\_VALUE [IRDs]** This read-only SQLINTEGER record field contains SQL\_TRUE if the column is an auto-incrementing column, or SQL\_FALSE if the column is not an auto-incrementing column. This field is read-only, but the underlying auto-incrementing column is not necessarily read-only.

**SQL\_DESC\_BASE\_COLUMN\_NAME [IRDs]** This read-only SQLCHAR record field contains the base column name for the result set column. If a base column name does not exist (as in the case of columns that are expressions), then this variable contains an empty string.

**SQL\_DESC\_BASE\_TABLE\_NAME [IRDs]** This read-only SQLCHAR record field contains the base table name for the result set column. If a base table name cannot be defined or is not applicable, then this variable contains an empty string.

**SQL\_DESC\_CASE\_SENSITIVE [Implementation descriptors]** This read-only SQLINTEGER record field contains SQL\_TRUE if the column or parameter is

## Descriptor FieldIdentifier argument values (CLI)

treated as case-sensitive for collations and comparisons, or `SQL_FALSE` if the column is not treated as case-sensitive for collations and comparisons, or if it is a non-character column.

**SQL\_DESC\_CATALOG\_NAME [IRDs]** This read-only `SQLCHAR` record field contains the catalog or qualifier name for the base table that contains the column. The return value is driver-dependent if the column is an expression or if the column is part of a view. If the data source does not support catalogs (or qualifiers) or the catalog or qualifier name cannot be determined, this variable contains an empty string.

**SQL\_DESC\_CONCISE\_TYPE [All]** This `SQLSMALLINT` header field specifies the concise data type for all data types.

The values in the `SQL_DESC_CONCISE_TYPE` and `SQL_DESC_TYPE` fields are interdependent. Each time one of the fields is set, the other must also be set. `SQL_DESC_CONCISE_TYPE` can be set by a call to `SQLBindCol()` or `SQLBindParameter()`, or `SQLSetDescField()`. `SQL_DESC_TYPE` can be set by a call to `SQLSetDescField()` or `SQLSetDescRec()`.

If `SQL_DESC_CONCISE_TYPE` is set to a concise data type, `SQL_DESC_TYPE` field is set to the same value, and the `SQL_DESC_DATETIME_INTERVAL_CODE` field is set to 0.

**SQL\_DESC\_DATA\_PTR [Application descriptors and IPDs]** This `SQLPOINTER` record field points to a variable that will contain the parameter value (for APDs) or the column value (for ARDs). The descriptor record (and either the column or parameter that it represents) is unbound if *TargetValuePtr* in a call to either `SQLBindCol()` or `SQLBindParameter()` is a null pointer, or the `SQL_DESC_DATA_PTR` field in a call to `SQLSetDescField()` or `SQLSetDescRec()` is set to a null pointer. Other fields are not affected if the `SQL_DESC_DATA_PTR` field is set to a null pointer. If the call to `SQLFetch()` or `SQLFetchScroll()` that fills in the buffer pointed to by this field did not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the buffer are undefined.

This field is a deferred field: it is not used at the time it is set, but is used at a later time by CLI to retrieve data.

Whenever the `SQL_DESC_DATA_PTR` field is set, CLI checks that the value in the `SQL_DESC_TYPE` field contains valid CLI or ODBC data types, and that all other fields affecting the data types are consistent. Refer to the consistency checks information for more detail.

**SQL\_DESC\_DATETIME\_INTERVAL\_CODE [All]** This `SQLSMALLINT` record field contains the subcode for the specific datetime data type when the `SQL_DESC_TYPE` field is `SQL_DATETIME`. This is true for both SQL and C data types.

This field can be set to the following for datetime data types:

*Table 168. Datetime subcodes*

Datetime types	DATETIME_INTERVAL_CODE
<code>SQL_TYPE_DATE/SQL_C_TYPE_DATE</code>	<code>SQL_CODE_DATE</code>
<code>SQL_TYPE_TIME/SQL_C_TYPE_TIME</code>	<code>SQL_CODE_TIME</code>

## Descriptor FieldIdentifier argument values (CLI)

Table 168. Datetime subcodes (continued)

Datetime types	DATETIME_INTERVAL_CODE
SQL_TYPE_TIMESTAMP/ SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

ODBC 3.0 defines other values (not listed here) for interval data types, which CLI does not support. If any other value is specified in a `SQLSetDescRec()` or `SQLSetDescField()` call, an error will be generated indicating HY092 (Option type out of range).

**SQL\_DESC\_DATETIME\_INTERVAL\_PRECISION [All]** ODBC 3.0 defines this `SQLINTEGER` record field, however, CLI does not support interval data types. The fixed value returned is 0. Any attempt to set this field will result in 01S02 (Option value changed).

**SQL\_DESC\_DISPLAY\_SIZE [IRDs]** This read-only `SQLINTEGER` record field contains the maximum number of characters required to display the data from the column. The value in this field is not the same as the descriptor field `SQL_DESC_LENGTH` because the `LENGTH` field is undefined for all numeric types.

**SQL\_DESC\_FIXED\_PREC\_SCALE [Implementation descriptors]** This read-only `SQLSMALLINT` record field is set to `SQL_TRUE` if the column is an exact numeric column and has a fixed precision and non-zero scale, or `SQL_FALSE` if the column is not an exact numeric column with a fixed precision and scale.

**SQL\_DESC\_INDICATOR\_PTR [Application descriptors]** In ARDs, this `SQLINTEGER *` record field points to the indicator variable. This variable contains `SQL_NULL_DATA` if the column value is `NULL`. For APDs, the indicator variable is set to `SQL_NULL_DATA` to specify `NULL` dynamic arguments. Otherwise, the variable is zero (unless the values in `SQL_DESC_INDICATOR_PTR` and `SQL_DESC_OCTET_LENGTH_PTR` are the same pointer).

If the `SQL_DESC_INDICATOR_PTR` field in an ARD is a null pointer, CLI is prevented from returning information about whether the column is `NULL` or not. If the column is `NULL` and `INDICATOR_PTR` is a null pointer, `SQLSTATE 22002`, Indicator variable required but not supplied, is returned when CLI attempts to populate the buffer after a call to `SQLFetch()` or `SQLFetchScroll()`. If the call to `SQLFetch()` or `SQLFetchScroll()` did not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the buffer are undefined.

The `SQL_DESC_INDICATOR_PTR` field determines whether the field pointed to by `SQL_DESC_OCTET_LENGTH_PTR` is set. If the data value for a column is `NULL`, CLI sets the indicator variable to `SQL_NULL_DATA`. The field pointed to by `SQL_DESC_OCTET_LENGTH_PTR` is then not set. If a `NULL` value is not encountered during the fetch, the buffer pointed to by `SQL_DESC_INDICATOR_PTR` is set to zero, and the buffer pointed to by `SQL_DESC_OCTET_LENGTH_PTR` is set to the length of the data.

If the `INDICATOR_PTR` field in an APD is a null pointer, the application cannot use this descriptor record to specify `NULL` arguments.

## Descriptor FieldIdentifier argument values (CLI)

This field is a deferred field: it is not used at the time it is set, but is used at a later time by CLI to store data.

**SQL\_DESC\_LABEL [IRDs]** This read-only SQLCHAR record field contains the column label or title. If the column does not have a label, this variable contains the column name. If the column is unnamed and unlabeled, this variable contains an empty string.

**SQL\_DESC\_LENGTH [All]** This SQLINTEGER record field is either the maximum or actual character length of a character string or a binary data type. It is the maximum character length for a fixed-length data type, or the actual character length for a variable-length data type. Its value always excludes the null termination character that ends the character string. Note that this field is a count of characters, not a count of bytes.

**SQL\_DESC\_LITERAL\_PREFIX [IRDs]** This read-only SQLCHAR record field contains the character or characters that CLI recognizes as a prefix for a literal of this data type. This variable contains an empty string for a data type for which a literal prefix is not applicable.

**SQL\_DESC\_LITERAL\_SUFFIX [IRDs]** This read-only SQLCHAR record field contains the character or characters that CLI recognizes as a suffix for a literal of this data type. This variable contains an empty string for a data type for which a literal suffix is not applicable.

**SQL\_DESC\_LOCAL\_TYPE\_NAME [Implementation descriptors]** This read-only SQLCHAR record field contains any localized (native language) name for the data type that may be different from the regular name of the data type. If there is no localized name, then an empty string is returned. This field is for display purposes only.

**SQL\_DESC\_NAME [Implementation descriptors]** This SQLCHAR record field in a row descriptor contains the column alias, if it applies. If the column alias does not apply, the column name is returned. In either case, the UNNAMED field is set to SQL\_NAMED. If there is no column name or a column alias, an empty string is returned in the NAME field and the UNNAMED field is set to SQL\_UNNAMED.

An application can set the SQL\_DESC\_NAME field of an IPD to a parameter name or alias to specify stored procedure parameters by name. The SQL\_DESC\_NAME field of an IRD is a read-only field; SQLSTATE HY091 (Invalid descriptor field identifier) will be returned if an application attempts to set it.

In IPDs, this field is undefined if dynamic parameters are not supported. If named parameters are supported and the version of CLI is capable of describing parameters, then the parameter name is returned in this field.

The column name value can be affected by the environment attribute SQL\_ATTR\_USE\_LIGHT\_OUTPUT\_SQLDA set by SQLSetEnvAttr().

**SQL\_DESC\_NULLABLE [Implementation descriptors]** In IRDs, this read-only SQLSMALLINT record field is SQL\_NULLABLE if the column can have NULL values; SQL\_NO\_NULLS if the column cannot have NULL values; or SQL\_NULLABLE\_UNKNOWN if it is not known whether the column accepts NULL values. This field pertains to the result set column, not the base column.

## Descriptor FieldIdentifier argument values (CLI)

In IPDs, this field is always set to `SQL_NULLABLE`, since dynamic parameters are always nullable, and cannot be set by an application.

**SQL\_DESC\_NUM\_PREC\_RADIX [All]** This `SQLINTEGER` field contains a value of 2 if the data type in the `SQL_DESC_TYPE` field is an approximate numeric data type, because the `SQL_DESC_PRECISION` field contains the number of bits. This field contains a value of 10 if the data type in the `SQL_DESC_TYPE` field is an exact numeric data type, because the `SQL_DESC_PRECISION` field contains the number of decimal digits. This field is set to 0 for all non-numeric data types.

**SQL\_DESC\_OCTET\_LENGTH [All]** This `SQLINTEGER` record field contains the length, in bytes, of a character string or binary data type. For fixed-length character types, this is the actual length in bytes. For variable-length character or binary types, this is the maximum length in bytes. This value always excludes space for the null termination character for implementation descriptors and always includes space for the null termination character for application descriptors. For application data, this field contains the size of the buffer. For APDs, this field is defined only for output or input/output parameters.

**SQL\_DESC\_OCTET\_LENGTH\_PTR [Application descriptors]** This `SQLINTEGER` \* record field points to a variable that will contain the total length in bytes of a dynamic argument (for parameter descriptors) or of a bound column value (for row descriptors).

For an APD, this value is ignored for all arguments except character string and binary; if this field points to `SQL_NTS`, the dynamic argument must be null-terminated. To indicate that a bound parameter will be a data-at-execute parameter, an application sets this field in the appropriate record of the APD to a variable that, at execute time, will contain the value `SQL_DATA_AT_EXEC`. If there is more than one such field, `SQL_DESC_DATA_PTR` can be set to a value uniquely identifying the parameter to help the application determine which parameter is being requested.

If the `OCTET_LENGTH_PTR` field of an ARD is a null pointer, CLI does not return length information for the column. If the `SQL_DESC_OCTET_LENGTH_PTR` field of an APD is a null pointer, CLI assumes that character strings and binary values are null terminated. (Binary values should not be null terminated, but should be given a length, in order to avoid truncation.)

If the call to `SQLFetch()` or `SQLFetchScroll()` that fills in the buffer pointed to by this field did not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the buffer are undefined.

This field is a deferred field: it is not used at the time it is set, but is used at a later time by CLI to buffer data.

By default this is a pointer to a 4-byte value.

**SQL\_DESC\_PARAMETER\_TYPE [IPDs]** This `SQLSMALLINT` record field is set to `SQL_PARAM_INPUT` for an input parameter, `SQL_PARAM_INPUT_OUTPUT` for an input/output parameter, or `SQL_PARAM_OUTPUT` for an output parameter. Set to `SQL_PARAM_INPUT` by default.

## Descriptor FieldIdentifier argument values (CLI)

For an IPD, the field is set to `SQL_PARAM_INPUT` by default if the IPD is not automatically populated by CLI (the `SQL_ATTR_ENABLE_AUTO_IPD` statement attribute is `SQL_FALSE`). An application should set this field in the IPD for parameters that are not input parameters.

**SQL\_DESC\_PRECISION [All]** This `SQLSMALLINT` record field contains the number of digits for an exact numeric type, the number of bits in the mantissa (binary precision) for an approximate numeric type, or the numbers of digits in the fractional seconds component for the `SQL_TYPE_TIME` or `SQL_TYPE_TIMESTAMP` data types. This field is undefined for all other data types.

**SQL\_DESC\_SCALE [All]** This `SQLSMALLINT` record field contains the defined scale for `DECIMAL` and `NUMERIC` data types. The field is undefined for all other data types.

**SQL\_DESC\_SCHEMA\_NAME [IRDs]** This read-only `SQLCHAR` record field contains the schema name of the base table that contains the column. For many DBMS's, this is the owner name. If the data source does not support schemas (or owners) or the schema name cannot be determined, this variable contains an empty string.

**SQL\_DESC\_SEARCHABLE [IRDs]** This read-only `SQLSMALLINT` record field is set to one of the following values:

- `SQL_PRED_NONE` if the column cannot be used in a `WHERE` clause. (This is the same as the `SQL_UNSEARCHABLE` value defined in ODBC 2.0.)
- `SQL_PRED_CHAR` if the column can be used in a `WHERE` clause, but only with the `LIKE` predicate. (This is the same as the `SQL_LIKE_ONLY` value defined in ODBC 2.0.)
- `SQL_PRED_BASIC` if the column can be used in a `WHERE` clause with all the comparison operators except `LIKE`. (This is the same as the `SQL_EXCEPT_LIKE` value defined in ODBC 2.0.)
- `SQL_PRED_SEARCHABLE` if the column can be used in a `WHERE` clause with any comparison operator.

**SQL\_DESC\_TABLE\_NAME [IRDs]** This read-only `SQLCHAR` record field contains the name of the base table that contains this column.

**SQL\_DESC\_TYPE [All]** This `SQLSMALLINT` record field specifies the concise SQL or C data type for all data types.

**Note:** ODBC 3.0 defines the `SQL_INTERVAL` data type which is not supported by CLI. Any behavior associated with this data type is not present in CLI.

The values in the `SQL_DESC_TYPE` and `SQL_DESC_CONCISE_TYPE` fields are interdependent. Each time one of the fields is set, the other must also be set. `SQL_DESC_TYPE` can be set by a call to `SQLSetDescField()` or `SQLSetDescRec()`. `SQL_DESC_CONCISE_TYPE` can be set by a call to `SQLBindCol()` or `SQLBindParameter()`, or `SQLSetDescField()`.

If `SQL_DESC_TYPE` is set to a concise data type, the `SQL_DESC_CONCISE_TYPE` field is set to the same value, and the `SQL_DESC_DATETIME_INTERVAL_CODE` field is set to 0.

## Descriptor FieldIdentifier argument values (CLI)

When the `SQL_DESC_TYPE` field is set by a call to `SQLSetDescField()`, the following fields are set to the following default values. The values of the remaining fields of the same record are undefined:

Table 169. Default values

<code>SQL_DESC_TYPE</code>	Other fields Implicitly Set
<code>SQL_CHAR</code> , <code>SQL_VARCHAR</code>	<code>SQL_DESC_LENGTH</code> is set to 1. <code>SQL_DESC_PRECISION</code> is set to 0.
<code>SQL_DECIMAL</code> , <code>SQL_NUMERIC</code>	<code>SQL_DESC_SCALE</code> is set to 0. <code>SQL_DESC_PRECISION</code> is set to the precision for the respective data type.
<code>SQL_FLOAT</code>	<code>SQL_DESC_PRECISION</code> is set to the default precision for <code>SQL_FLOAT</code> .
<code>SQL_DATETIME</code>	<code>SQL_DESC_CONCISE_TYPE</code> and/or <code>SQL_DESC_DATETIME_INTERVAL_CODE</code> may be set implicitly to indicate a DATE SQL or C type.
<code>SQL_INTERVAL</code>	This data type is not supported by CLI.

When an application calls `SQLSetDescField()` to set fields of a descriptor, rather than calling `SQLSetDescRec()`, the application must first declare the data type. If the values implicitly set are unacceptable, the application can then call `SQLSetDescField()` to set the unacceptable value explicitly.

**SQL\_DESC\_TYPE\_NAME [Implementation descriptors]** This read-only `SQLCHAR` record field contains the data-source-dependent type name (for example, `CHAR`, `VARCHAR`, and so on). If the data type name is unknown, this variable contains an empty string.

**SQL\_DESC\_UNNAMED [Implementation descriptors]** This `SQLSMALLINT` record field in a row descriptor is set to either `SQL_NAMED` or `SQL_UNNAMED`. If the `NAME` field contains a column alias, or if the column alias does not apply, the `UNNAMED` field is set to `SQL_NAMED`. If there is no column name or a column alias, the `UNNAMED` field is set to `SQL_UNNAMED`.

An application can set the `SQL_DESC_UNNAMED` field of an IPD to `SQL_UNNAMED`. `SQLSTATE HY091 (Invalid descriptor field identifier)` is returned if an application attempts to set the `SQL_DESC_UNNAMED` field of an IPD to `SQL_NAMED`. The `SQL_DESC_UNNAMED` field of an IRD is read-only; `SQLSTATE HY091 (Invalid descriptor field identifier)` will be returned if an application attempts to set it.

**SQL\_DESC\_UNSIGNED [Implementation descriptors]** This read-only `SQLSMALLINT` record field is set to `SQL_TRUE` if the column type is unsigned or non-numeric, or `SQL_FALSE` if the column type is signed.

**SQL\_DESC\_UPDATABLE [IRDs]** This read-only `SQLSMALLINT` record field is set to one of the following values:

- `SQL_ATTR_READONLY` if the result set column is read-only.
- `SQL_ATTR_WRITE` if the result set column is read-write.
- `SQL_ATTR_READWRITE_UNKNOWN` if it is not known whether the result set column is updatable or not.

`SQL_DESC_UPDATABLE` describes the updatability of the column in the result set, not the column in the base table. The updatability of the column in the base table

## Descriptor FieldIdentifier argument values (CLI)

on which this result set column is based may be different than the value in this field. Whether a column is updatable can be based on the data type, user privileges, and the definition of the result set itself. If it is unclear whether a column is updatable, `SQL_UPDT_READWRITE_UNKNOWN` should be returned.

**SQL\_DESC\_USER\_DEFINED\_TYPE\_CODE [IRDS]** This read-only `SQLINTEGER` returns information that describes the nature of a column's data type. Four values may be returned:

- `SQL_TYPE_BASE`: the column data type is a base data type, such as `CHAR`, `DATE`, or `DOUBLE`.
- `SQL_TYPE_DISTINCT`: the column data type is a distinct user-defined type.
- `SQL_TYPE_REFERENCE`: the column data type is a reference user-defined type.
- `SQL_TYPE_STRUCTURED`: the column data type is a structured user-defined type.

**SQL\_DESC\_CARDINALITY [APD, IPD]** This `SQLLEN` record field indicates the maximum cardinality of the array value for the specified parameter marker. The `IPD` value indicates the maximum cardinality that can be sent to the database for an array value. The `APD` value indicates the maximum cardinality for an array value which is how the application indicates the maximum amount of allocated storage for output parameter array values for a `CALL` statement.

**SQL\_DESC\_CARDINALITY\_PTR [APD]** This `SQLLEN *` record field points to a variable that will contain the cardinality of a parameter when the statement is executed. For input parameter markers, this is how the application provides the actual cardinality of an array value. For output parameter markers, this is the location where CLI will indicate the cardinality of the returned array value. Note that for output parameters, this value indicates the cardinality of the array returned by the stored procedure - not necessarily the cardinality written to the application, since it's possible for `SQL_DESC_CARDINALITY(APD)` to be less than the actual cardinality returned from the procedure. This is a deferred field - it is not used at the time it is set, but is used at a later time by DB2 CLI.

---

## Descriptor header and record field initialization values (CLI)

The following tables list the initialization of each field for each type of descriptor, with `D` indicating that the field is initialized with a default, and `ND` indicating that the field is initialized without a default. If a number is shown, the default value of the field is that number. The tables also indicate whether a field is read/write (`R/W`) or read-only (`R`).

The initialization of header fields is as follows:



## Descriptor header and record field initialization values (CLI)

Table 170. Initialization of header fields

Descriptor header field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_ALLOC_TYPE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R</li> <li>• APD: R</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit</li> <li>• APD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit</li> <li>• IRD: SQL_DESC_ALLOC_AUTO</li> <li>• IPD: SQL_DESC_ALLOC_AUTO</li> </ul>
SQL_DESC_ARRAY_SIZE	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: <sup>a</sup></li> <li>• APD: <sup>a</sup></li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT *	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R/W</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Null ptr</li> <li>• APD: Null ptr</li> <li>• IRD: Null ptr</li> <li>• IPD: Null ptr</li> </ul>
SQL_DESC_BIND_OFFSET_PTR	SQLINTEGER *	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Null ptr</li> <li>• APD: Null ptr</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_BIND_TYPE	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: SQL_BIND_BY_COLUMN</li> <li>• APD: SQL_BIND_BY_COLUMN</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_COUNT	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: 0</li> <li>• APD: 0</li> <li>• IRD: D</li> <li>• IPD: 0</li> </ul>
SQL_DESC_ROWS_PROCESSED_PTR	SQLINTEGER *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R/W</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: Null Ptr</li> <li>• IPD: Null Ptr</li> </ul>

- a** These fields are defined only when the IPD is automatically populated by CLI. If the fields are not automatically populated then they are undefined. If an application attempts to set these fields, SQLSTATE HY091 (Invalid descriptor field identifier.) will be returned.

The initialization of record fields is as follows:

Table 171. Initialization of record fields

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>

## Descriptor header and record field initialization values (CLI)

Table 171. Initialization of record fields (continued)

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: D <sup>a</sup></li> </ul>
SQL_DESC_CATALOG_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: SQL_C_DEFAULT</li> <li>• APD: SQL_C_DEFAULT</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_DATA_PTR	SQLPOINTER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Null ptr</li> <li>• APD: Null ptr</li> <li>• IRD: Unused</li> <li>• IPD: Unused <sup>b</sup></li> </ul>
SQL_DESC_DATETIME_INTERVAL_CODE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_DISPLAY_SIZE	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: D <sup>a</sup></li> </ul>
SQL_DESC_INDICATOR_PTR	SQLINTEGER *	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Null ptr</li> <li>• APD: Null ptr</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_LABEL	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>

## Descriptor header and record field initialization values (CLI)

Table 171. Initialization of record fields (continued)

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_LENGTH	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: D <sup>a</sup></li> </ul>
SQL_DESC_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_NULLABLE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: N</li> <li>• IPD: ND</li> </ul>
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_OCTET_LENGTH	SQLINTEGER	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_OCTET_LENGTH_PTR	SQLINTEGER *	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Null ptr</li> <li>• APD: Null ptr</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IPD: Unused</li> <li>• IRD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IPD: Unused</li> <li>• IRD: D=SQL_PARAM_INPUT</li> </ul>
SQL_DESC_PRECISION	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_SCALE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>

## Descriptor header and record field initialization values (CLI)

Table 171. Initialization of record fields (continued)

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_SCHEMA_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_SEARCHABLE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_TABLE_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_TYPE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: R/W</li> <li>• APD: R/W</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: SQL_C_DEFAULT</li> <li>• APD: SQL_C_DEFAULT</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_TYPE_NAME	SQLCHAR *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: D <sup>a</sup></li> </ul>
SQL_DESC_UNNAMED	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: ND</li> <li>• APD: ND</li> <li>• IRD: D</li> <li>• IPD: ND</li> </ul>
SQL_DESC_UNSIGNED	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: R</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: D <sup>a</sup></li> </ul>
SQL_DESC_UPDATABLE	SQLSMALLINT	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: R</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: Unused</li> <li>• IRD: D</li> <li>• IPD: Unused</li> </ul>
SQL_DESC_CARDINALITY	SQLLEN	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: R/W</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: D</li> <li>• IRD: Unused</li> <li>• IPD: D</li> </ul>
SQL_DESC_CARDINALITY_PTR	SQLLEN *	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: R/W</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>	<ul style="list-style-type: none"> <li>• ARD: Unused</li> <li>• APD: D</li> <li>• IRD: Unused</li> <li>• IPD: Unused</li> </ul>

- a** These fields are defined only when the IPD is automatically populated by CLI. If the fields are not automatically populated then they are undefined. If an application attempts to set these fields, SQLSTATE HY091 (Invalid descriptor field identifier.) will be returned.
- b** The SQL\_DESC\_DATA\_PTR field in the IPD can be set to force a consistency check. In a subsequent call to SQLGetDescField() or SQLGetDescRec(), CLI is not required to return the value that SQL\_DESC\_DATA\_PTR was set to.

## Chapter 6. Header and record fields for the DiagIdentifier argument (CLI)

### Header fields

You can specify the following header fields for the *DiagIdentifier* argument. The only diagnostic header fields that you can define for a descriptor field are SQL\_DIAG\_NUMBER and SQL\_DIAG\_RETURNCODE.

Table 172. Header fields for the DiagIdentifier argument

Header field	Return type	Description
SQL_DIAG_CURSOR_ROW_COUNT	SQLINTEGER	<p>The count of rows in the cursor. The semantics of the field depend upon the SQLGetInfo() information types, which indicate which row counts are available for each cursor type (in the SQL_CA2_CRC_EXACT and SQL_CA2_CRC_APPROXIMATE bits):</p> <ul style="list-style-type: none"> <li>• SQL_DYNAMIC_CURSOR_ATTRIBUTES2</li> <li>• SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2</li> <li>• SQL_KEYSET_CURSOR_ATTRIBUTES2</li> <li>• SQL_STATIC_CURSOR_ATTRIBUTES2</li> </ul> <p>A value is defined in this field only for statement handles and only after a call to the SQLExecute(), SQLExecDirect(), or SQLMoreResults() function.</p> <p>Calling the SQLGetDiagField() function with a <i>DiagIdentifier</i> argument value of SQL_DIAG_CURSOR_ROW_COUNT on a handle other than a statement handle returns SQL_ERROR.</p>
SQL_DIAG_DYNAMIC_FUNCTION	CHAR *	<p>A string that describes the SQL statement that the underlying function executed (for the values that CLI supports, see Dynamic function fields). A value is defined in this field only for statement handles, and only after a call to the SQLExecute(), SQLExecDirect(), or SQLMoreResults() function.</p>

## Header and record fields for the DiagIdentifier argument (CLI)

Table 172. Header fields for the DiagIdentifier argument (continued)

Header field	Return type	Description
SQL_DIAG_DYNAMIC_FUNCTION_CODE		<p>A numeric code that describes the SQL statement that was executed by the underlying function (for the values that CLI supports, see Dynamic function fields). A value is defined in this field only for statement handles, and only after a call to the SQLExecute(), SQLExecDirect(), or SQLMoreResults() function.</p> <p>Calling the SQLGetDiagField() function with a <i>DiagIdentifier</i> argument value of SQL_DIAG_DYNAMIC_FUNCTION_CODE on a handle other than a statement handle returns SQL_ERROR.</p>
SQL_DIAG_NETWORK_STATISTICS	SQLINTEGER	<p>A structure containing network statistics for a connection.</p> <p>The statistics include the following:</p> <ul style="list-style-type: none"> <li>• Database processing time in microseconds</li> <li>• Network time (including database processing time) in microseconds</li> <li>• Number of bytes that are sent to the database server</li> <li>• Number of bytes that are received from the database server</li> <li>• Number of DRDA round trips</li> </ul> <p>This field is available starting in Version 9.7, Fix Pack 3.</p> <p>CLI accumulates statistics for a connection when the SQL_ATTR_NETWORK_STATISTICS connection attribute is enabled. After an application calls the SQLGetDiagField() function to retrieve the statistics, CLI resets the internal counters that the connection uses to accumulate the statistics. A value is defined in this field only for connection handles, SQL_HANDLE_DBC.</p> <ul style="list-style-type: none"> <li>• Calling the SQLGetDiagField() function with a <i>DiagIdentifier</i> argument of SQL_DIAG_NETWORK_STATISTICS with an SQL_HANDLE_ENV returns ERROR.</li> <li>• Calling the SQLGetDiagField() function with a <i>DiagIdentifier</i> argument of SQL_DIAG_NETWORK_STATISTICS and a SQL_HANDLE_STMT or SQL_HANDLE_DESC returns the diagnostic identifier field for the underlying connection handle associated with the specified statement or descriptor handle.</li> </ul>

## Header and record fields for the DiagIdentifier argument (CLI)

Table 172. Header fields for the DiagIdentifier argument (continued)

Header field	Return type	Description
SQL_DIAG_NUMBER	SQL_NET_STATS <b>Note:</b> The SQL_NET_STATS structure is defined in the sqlcli1.h file.	The number of status records that are available for the specified handle.
SQL_DIAG_RELATIVE_COST_ESTIMATE	SQLINTEGER	A relative cost estimate of the resources that are required to process a statement if the SQLPrepare() function is invoked and successful. If deferred prepare is enabled, this field has the value 0 until the statement is executed.
SQL_DIAG_RETURNCODE	RETCODE	The return code of the last executed function that is associated with the specified handle. If a function has not been called on the <i>Handle</i> , SQL_SUCCESS is returned in the SQL_DIAG_RETURNCODE field.
SQL_DIAG_ROW_COUNT	SQLINTEGER	The number of rows that are affected by an insert, delete, or update performed by the SQLExecute(), SQLExecDirect(), or SQLSetPos() function. The value of this field is defined after a cursor specification has been executed and only for statement handles. The data in this field is returned in the <i>RowCountPtr</i> argument of the SQLRowCount() function. The data in this field is reset after every function call, whereas the row count returned by the SQLRowCount() function remains the same until the statement state is reset to the prepared or allocated state.
SQL_DIAG_TOLERATED_ERROR		<p>Beginning with WebSphere® Federated Server V9.1, you can specify an Error Tolerant Nested Table Expression (ETNTE). This allows you to specify errors that can be tolerated and is of particular use when one or more data sources are unavailable during a UNION ALL query. By tolerating the absence of one of the unions (because the source is offline) a result set can still be processed and SQLFetch can be called repeatedly until it returns <b>SQL_NO_DATA_FOUND</b>(100).</p> <p>If tolerate errors are encountered then a CLI application can discover this by doing one of two things:</p> <ul style="list-style-type: none"> <li>• Call the SQLGetDiagField() function with a <b>DiagIdentifier</b> argument value of SQL_DIAG_TOLERATED_ERROR. If a tolerated error was processed, TRUE is returned. Otherwise, FALSE is returned.</li> <li>• Call the SQLGetDiagRec() function. If a tolerated error was processed, the diagnostic record displays error SQL20383W. Otherwise, error SQL0100W is displayed.</li> </ul>

## Header and record fields for the *DiagIdentifier* argument (CLI)

### Record fields

You can specify the following record fields for the *DiagIdentifier* argument.

Table 173. Record fields for the *DiagIdentifier* argument

Record field	Return type	Description
SQL_DIAG_CLASS_ORIGIN	CHAR *	A string that indicates the document that defines the class and subclass portions of the SQLSTATE value in a record.  CLI always returns an empty string in the SQL_DIAG_CLASS_ORIGIN field.
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	A value that represents the column number in a result set if the value of the SQL_DIAG_ROW_NUMBER field is a valid row number in a rowset or set of parameters. Result set column numbers always start at 1; if the status record pertains to a bookmark column, the value of the field can be zero. The field has the value SQL_NO_COLUMN_NUMBER if the status record is not associated with a column number. If CLI cannot determine the column number that a record is associated with, this field has the value SQL_COLUMN_NUMBER_UNKNOWN. A value is defined in this field only for statement handles.
SQL_DIAG_CONNECTION_NAME	CHAR *	A string that indicates the name of the connection that a diagnostic record relates to.  CLI always returns an empty string in the SQL_DIAG_CONNECTION_NAME field.
SQL_DIAG_ERRMC	CHAR *	A string containing one or more message tokens that are separated by X'FF'.
SQL_DIAG_MESSAGE_TEXT	CHAR *	An informational message about an error or warning.
SQL_DIAG_NATIVE	SQLINTEGER	A native error code that is specific to a driver or data source, if a code exists; otherwise, the driver returns 0.
SQL_DIAG_ROW_NUMBER	SQLINTEGER	The row number in the rowset, or the parameter number in the set of parameters, with which the status record is associated. This field has the value SQL_NO_ROW_NUMBER if the status record is not associated with a row number. If CLI cannot determine the row number that a record is associated with, this field has the value SQL_ROW_NUMBER_UNKNOWN. A value is defined in this field only for statement handles.



## Header and record fields for the DiagIdentifier argument (CLI)

Table 173. Record fields for the DiagIdentifier argument (continued)

Record field	Return type	Description
SQL_DIAG_SERVER_NAME	CHAR *	A string that indicates the server name that a diagnostic record relates to. The string is the same as the value returned for a call to the SQLGetInfo() function with an <i>InfoType</i> argument value of SQL_DATA_SOURCE_NAME. For diagnostic data structures associated with an environment handle and for diagnostics that do not relate to any server, this field is a 0-length string.
SQL_DIAG_SQLSTATE	CHAR *	A 5-character SQLSTATE diagnostic code.
SQL_DIAG_SUBCLASS_ORIGIN	CHAR *	A string, with the same format and valid values as the SQL_DIAG_CLASS_ORIGIN field, that identifies the defining portion of the subclass portion of the SQLSTATE code.

### Values of the dynamic function fields

The following table describes the values of SQL\_DIAG\_DYNAMIC\_FUNCTION and SQL\_DIAG\_DYNAMIC\_FUNCTION\_CODE that apply to each type of SQL statement that is executed by a call to the SQLExecute() or SQLExecDirect() function. CLI uses the values in this table; ODBC specifies other values.

Table 174. Values of dynamic function fields

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
alter-table-statement	ALTER TABLE	SQL_DIAG_ALTER_TABLE
create-index-statement	CREATE INDEX	SQL_DIAG_CREATE_INDEX
create-table-statement	CREATE TABLE	SQL_DIAG_CREATE_TABLE
create-view-statement	CREATE VIEW	SQL_DIAG_CREATE_VIEW
cursor-specification	SELECT CURSOR	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	DYNAMIC DELETE CURSOR	SQL_DIAG_DYNAMIC_DELETE_CURSOR
delete-statement-searched	DELETE WHERE	SQL_DIAG_DELETE_WHERE
drop-index-statement	DROP INDEX	SQL_DIAG_DROP_INDEX
drop-table-statement	DROP TABLE	SQL_DIAG_DROP_TABLE
drop-view-statement	DROP VIEW	SQL_DIAG_DROP_VIEW
grant-statement	GRANT	SQL_DIAG_GRANT
insert-statement	INSERT	SQL_DIAG_INSERT
ODBC-procedure-extension	CALL	SQL_DIAG_PROCEDURE_CALL
revoke-statement	REVOKE	SQL_DIAG_REVOKE
update-statement-positioned	DYNAMIC UPDATE CURSOR	SQL_DIAG_DYNAMIC_UPDATE_CURSOR
update--statement-searched	UPDATE WHERE	SQL_DIAG_UPDATE_WHERE
merge-statement	MERGE	SQL_DIAG_MERGE

## Header and record fields for the DiagIdentifier argument (CLI)

Table 174. Values of dynamic function fields (continued)

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
Unknown	empty string	SQL_DIAG_UNKNOWN_STATEMENT

---

## Chapter 7. CLI data type attributes

---

### SQL symbolic and default data types for CLI applications

The table lists each of the SQL data types that are used by Call Level Interface (CLI) applications, with its corresponding symbolic name, and the default C symbolic name.

#### SQL data type

This column contains the SQL data types as they would display in an SQL CREATE statement. The SQL data types are dependent on the DBMS.

#### Symbolic SQL data type

This column contains SQL symbolic names that are defined (in `sqlcli.h`) as an integer value. Various functions use these values to identify the SQL data types that are listed in the first column.

#### Default C symbolic data type

This column contains C symbolic names, which are also defined as integer values. These values are used in various function arguments to identify the C data type. Various functions use the symbolic names, such as `SQLBindParameter()`, `SQLGetData()`, and `SQLBindCol()`, to indicate the C data types of the application variables. Instead of explicitly identifying C data types when calling these functions, you can specify `SQL_C_DEFAULT` instead, and CLI assumes a default C data type based on the SQL data type of the parameter or column as shown by the following table. For example, the default C data type of `SQL_DECIMAL` is `SQL_C_CHAR`.

Applications should not use the `SQL_C_DEFAULT` data type to define C data types because it is less efficient for CLI. Explicitly indicating the C data type in the application is preferred, because it yields better performance than using `SQL_C_DEFAULT`.

Table 175. SQL symbolic and default data types

SQL data type	Symbolic SQL data type	Default symbolic C data type
BIGINT	SQL_BIGINT	SQL_C_SBIGINT
BINARY <sup>f</sup>	SQL_BINARY	SQL_C_BINARY
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR <sup>a</sup>	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
BOOLEAN <sup>d</sup>	SQL_BOOLEAN	SQL_C_DEFAULT
CHAR	SQL_CHAR	SQL_C_CHAR
CHAR	SQL_TINYINT	SQL_C_TINYINT
CHAR FOR BIT DATA <sup>b</sup>	SQL_BINARY	SQL_C_BINARY
CHAR FOR BIT DATA	SQL_BIT	SQL_C_BINARY
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR <sup>a</sup>	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
CURSOR <sup>d</sup>	SQL_CURSORHANDLE	SQL_C_CURSORHANDLE
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR
DBCLOB LOCATOR <sup>a</sup>	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DECFLOAT(16)	SQL_DECFLOAT	SQL_C_CHAR

## SQL symbolic and default data types for CLI applications

Table 175. SQL symbolic and default data types (continued)

SQL data type	Symbolic SQL data type	Default symbolic C data type
DECFLOAT(34)	SQL_DECFLOAT	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR <sup>b</sup>	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA <sup>b</sup>	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC <sup>b</sup>	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
LONG VARGRAPHIC <sup>b</sup>	SQL_WLONGVARCHAR	SQL_C_DBCHAR
NUMERIC <sup>c</sup>	SQL_NUMERIC <sup>c</sup>	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
ROW <sup>d</sup>	SQL_ROW	SQL_C_DEFAULT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
TIMESTAMP WITH TIMEZONE	SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_C_TYPE_TIMESTAMP_EXT_TZ
VARBINARY <sup>f</sup>	SQL_VARBINARY	SQL_C_BINARY
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA <sup>b</sup>	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR
VARGRAPHIC	SQL_WVARIABLE	SQL_C_DBCHAR
WCHAR	SQL_WCHAR	SQL_C_WCHAR
XML <sup>e</sup>	SQL_XML	SQL_C_BINARY

- **a** LOB locator types are not persistent SQL data types. Columns cannot be defined with a locator type as they are only used to describe parameter markers, or to represent a LOB value.
- **b** LONG data types and FOR BIT DATA data types should be replaced by an appropriate LOB type whenever possible.
- **c** NUMERIC is a synonym for DECIMAL on DB2 for z/OS (Version 9.1 and later), DB2 Server for VSE & VM and DB2 Database for Linux, UNIX, and Windows.
- **d** BOOLEAN, CURSOR, and ROW types are only supported to provide applications with correct DESCRIBE information for database table columns or procedure parameters. No bind-in or bind-out is supported for these types. These types are recognized by the following CLI/ODBC APIs only: SQLExtendedDescribe() and SQLDescribeParam().
- **e** Starting in DB2 Version 9.7 Fix Pack 5, the SQL\_XML data type is supported for DB2 for i Version 7 Release 1 servers or later releases.
- **f** Starting in DB2 Version 9.7 Fix Pack 6, SQL\_BINARY and SQL\_VARBINARY data types are supported for DB2 for i Version 6 Release 1 servers or later releases.

**Note:**

You cannot transfer the data types DATE, DECIMAL, DECFLOAT(16), DECFLOAT(34), NUMERIC, TIME, and TIMESTAMP to their default C buffer types without a conversion.

## C data types for CLI applications

The following table lists the generic type definitions for each symbolic C type that is used in CLI applications.

**C symbolic data type**

This column contains C symbolic names, defined as integer values. These values are used in various function arguments to identify the C data type shown in the last column.

**C type**

This column contains C defined types, defined in `sqlcli.h` using a C typedef statement. The values in this column should be used to declare all CLI related variables and arguments, in order to make the application more portable. Refer to Table 178 on page 516 for a list of additional symbolic data types used for function arguments.

**Base C type**

This column is shown for reference only. All variables and arguments should be defined using the symbolic types in the previous column since the base C type is platform dependent. Some of the values are C structures that are described in Table 177 on page 514.

*Table 176. C data types*

<b>C symbolic data type</b>	<b>C type</b>	<b>Base C type</b>
SQL_C_BINARY	SQLCHAR	unsigned char
SQL_C_BINARYXML	SQLCHAR	unsigned char
SQL_C_BIT	SQLCHAR	unsigned char or char (Value 1 or 0)
SQL_C_BLOB_LOCATOR <sup>a</sup>	SQLINTEGER	32-bit integer
SQL_C_CLOB_LOCATOR <sup>a</sup>	SQLINTEGER	32-bit integer
SQL_C_CHAR	SQLCHAR	unsigned char
SQL_C_DBCHAR	SQLDBCHAR	wchar_t
SQL_C_DBCLOB_LOCATOR	SQLINTEGER	32-bit integer
SQL_C_DECIMAL64	SQLDECIMAL64	see Table 177 on page 514
SQL_C_DECIMAL128	SQLDECIMAL128	see Table 177 on page 514
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_FLOAT	SQLREAL	float
SQL_C_LONG	SQLINTEGER	32-bit integer
SQL_C_NUMERIC <sup>b</sup>	SQL_NUMERIC_STRUCT	see Table 177 on page 514
SQL_C_SBIGINT	SQLBIGINT	64-bit integer
SQL_C_SHORT	SQLSMALLINT	16-bit integer
SQL_C_TINYINT	SQLSCHAR	signed char (Range -128 to 127)
SQL_C_TYPE_DATE	DATE_STRUCT	see Table 177 on page 514
SQL_C_TYPE_TIME	TIME_STRUCT	see Table 177 on page 514
SQL_C_TYPE_TIMESTAMP	TIMESTAMP_STRUCT	see Table 177 on page 514
SQL_C_TYPE_TIMESTAMP_EXT	TIMESTAMP_STRUCT_EXT	see Table 177 on page 514
SQL_C_TYPE_TIMESTAMP_EXT_TZ	TIMESTAMP_STRUCT_EXT_TZ	see Table 177 on page 514
SQL_C_UBIGINT	SQLUBIGINT	unsigned 64-bit integer
SQL_C_ULONG	SQLUINTEGER	unsigned 32-bit integer
SQL_C_USHORT	SQLUSMALLINT	unsigned 16-bit integer
SQL_C_UTINYINT	SQLUCHAR	unsigned char
SQL_C_WCHAR	SQLWCHAR	wchar_t

## C data types for CLI applications

Table 176. C data types (continued)

C symbolic data type	C type	Base C type
	<ul style="list-style-type: none"> <li>• a LOB Locator Types.</li> <li>• b Windows only.</li> </ul>	
<p><b>Note:</b> SQL file reference data types (used in embedded SQL) are not needed in CLI.</p> <p><b>Note:</b> You use the SQL_C_BINARYXML C data type with the binary XML data transmission format, which is supported by DB2 CLI starting in DB2 z/OS Version 9.7 Fixpack 1, and in DB2 LUW Version 9.7 Fixpack 5. The DB2 server must also be at a level that supports the binary XML format.</p>		

Table 177. C structures

C type	Generic structure	Windows structure
DATE_STRUCT	<pre>typedef struct DATE_STRUCT {     SQLSMALLINT    year;     SQLUSMALLINT   month;     SQLUSMALLINT   day; } DATE_STRUCT;</pre>	<pre>typedef struct tagDATE_STRUCT {     SWORD    year;     UWORD   month;     UWORD   day; } DATE_STRUCT;</pre>
TIME_STRUCT	<pre>typedef struct TIME_STRUCT {     SQLUSMALLINT   hour;     SQLUSMALLINT   minute;     SQLUSMALLINT   second; } TIME_STRUCT;</pre>	<pre>typedef struct tagTIME_STRUCT {     UWORD   hour;     UWORD   minute;     UWORD   second; } TIME_STRUCT;</pre>
TIMESTAMP_STRUCT	<pre>typedef struct TIMESTAMP_STRUCT {     SQLUSMALLINT   year;     SQLUSMALLINT   month;     SQLUSMALLINT   day;     SQLUSMALLINT   hour;     SQLUSMALLINT   minute;     SQLUSMALLINT   second;     SQLINTEGER     fraction; } TIMESTAMP_STRUCT;</pre>	<pre>typedef struct tagTIMESTAMP_STRUCT {     SWORD    year;     UWORD   month;     UWORD   day;     UWORD   hour;     UWORD   minute;     UWORD   second;     UDWORD  fraction; } TIMESTAMP_STRUCT;</pre>
TIMESTAMP_STRUCT_EXT	<pre>typedef struct TIMESTAMP_STRUCT_EXT {     SQLSMALLINT   year;     SQLUSMALLINT   month;     SQLUSMALLINT   day;     SQLUSMALLINT   hour;     SQLUSMALLINT   minute;     SQLUSMALLINT   second;     SQLINTEGER     fraction;     /* 1-9 digits */     SQLINTEGER     fraction2;     /* 10-12 digits */ } TIMESTAMP_STRUCT_EXT;</pre>	(No Windows structure. Only a generic structure.)

Table 177. C structures (continued)

C type	Generic structure	Windows structure
TIMESTAMP_STRUCT_EXT_TZ	<pre>typedef struct TIMESTAMP_STRUCT {     SQLSMALLINT    year;     SQLUSMALLINT   month;     SQLUSMALLINT   day;     SQLUSMALLINT   hour;     SQLUSMALLINT   minute;     SQLUSMALLINT   second;     SQLINTEGER      fraction;     SQLINTEGER      fraction2;     SQLSMALLINT    timezone_hour; /*-12 to 14*/     SQLUSMALLINT   timezone_minute; /*-59 to 59*/ } TIMESTAMP_STRUCT_EXT_TZ;</pre>	(No Windows structure. Only a generic structure.)
SQLDECIMAL64	<pre>typedef struct tagSQLDECIMAL64 {     union {         SQLDOUBLE dummy;         SQLCHAR dec64             [SQL_DECFLOAT16_                 COEFFICIENT_LEN];     } udec64; } SQLDECIMAL64;</pre>	(No Windows structure. Only a generic structure.)
SQLDECIMAL128	<pre>typedef struct tagSQLDECIMAL128 {     union {         SQLDOUBLE dummy;         SQLCHAR dec128             [SQL_DECFLOAT34_                 COEFFICIENT_LEN];     } udec128; } SQLDECIMAL128;</pre>	(No Windows structure. Only a generic structure.)
SQL_NUMERIC_STRUCT	(No generic structure. Only a Windows structure.)	<pre>typedef struct tagSQL_NUMERIC_STRUCT {     SQLCHAR    precision;     SQLCHAR    scale;     SQLCHAR    sign;<sup>a</sup>     SQLCHAR         val[SQL_MAX_NUMERIC_LEN];<sup>b c</sup> } SQL_NUMERIC_STRUCT;</pre>

Refer to Table 178 on page 516 for more information on the SQLUSMALLINT C data type.

- **a** Sign field: 1 = positive, 2 = negative
- **b** A number is stored in the val field of the SQL\_NUMERIC\_STRUCT structure as a scaled integer, in little endian mode (the leftmost byte being the least-significant byte). For example, the number 10.001 base 10, with a scale of 4, is scaled to an integer of 100010. Because this is 186AA in hexadecimal format, the value in SQL\_NUMERIC\_STRUCT would be “AA 86 01 00 00 ... 00”, with the number of bytes defined by the SQL\_MAX\_NUMERIC\_LEN #define.
- **c** The precision and scale fields of the SQL\_C\_NUMERIC data type are never used for input from an application, only for output from the driver to the application. When the driver writes a numeric value into the SQL\_NUMERIC\_STRUCT, it will use its own default as the value for the precision field, and it will use the value in the SQL\_DESC\_SCALE field of the application descriptor (which defaults to 0) for the scale field. An application can provide its own values for precision and scale by setting the SQL\_DESC\_PRECISION and SQL\_DESC\_SCALE fields of the application descriptor.

## C data types for CLI applications

As well as the data types that map to SQL data types, there are also C symbolic types used for other function arguments such as pointers and handles. Both the generic and ODBC data types are shown below.

**Note:** There are two kinds of drivers that ship with the product: the CLI driver, and the 64-bit ODBC driver. The 64-bit ODBC Driver handles the differences with type definitions between various ODBC Managers.

Table 178. C Data types and base C data types

Defined C type	Base C type	Typical usage
SQLPOINTER	void *	Pointer to storage for data and parameters.
SQLHANDLE	<ol style="list-style-type: none"> <li>1. void *</li> <li>2. 32-bit integer</li> </ol>	Handle used to reference all 4 types of handle information. <ol style="list-style-type: none"> <li>1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver</li> <li>2. 32-bit value for all 32-bit platforms and 64-bit CLI Drivers</li> </ol>
SQLHENV	<ol style="list-style-type: none"> <li>1. void *</li> <li>2. 32-bit integer</li> </ol>	Handle referencing environment information. <ol style="list-style-type: none"> <li>1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver</li> <li>2. 32-bit value for all 32-bit platforms and 64-bit CLI Drivers</li> </ol>
SQLHDBC	<ol style="list-style-type: none"> <li>1. void *</li> <li>2. 32-bit integer</li> </ol>	Handle referencing database connection information. <ol style="list-style-type: none"> <li>1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver</li> <li>2. 32-bit value for all 32-bit platforms and 64-bit CLI Drivers</li> </ol>
SQLHSTMT	<ol style="list-style-type: none"> <li>1. void *</li> <li>2. 32-bit integer</li> </ol>	Handle referencing statement information. <ol style="list-style-type: none"> <li>1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver</li> <li>2. 32-bit value for all 32-bit platforms and 64-bit CLI Drivers</li> </ol>
SQLUSMALLINT	unsigned 16-bit integer	Function input argument for unsigned short integer values.
SQLINTEGER	unsigned 32-bit integer	Function input argument for unsigned long integer values.
SQLRETURN	16-bit integer	Return code from CLI functions.
SQLULEN	<ol style="list-style-type: none"> <li>1. unsigned 64-bit integer</li> <li>2. unsigned 32-bit integer</li> </ol>	<ol style="list-style-type: none"> <li>1. Function input or output argument for unsigned 64-bit integer values (64-bit ODBC driver).</li> <li>2. Function input or output argument for unsigned 32-bit integer values (all other drivers).</li> </ol>
SQLLEN	<ol style="list-style-type: none"> <li>1. 64-bit integer</li> <li>2. 32-bit integer</li> </ol>	<ol style="list-style-type: none"> <li>1. Function input or output argument for 64-bit integer values (64-bit ODBC driver).</li> <li>2. Function input or output argument for 32-bit integer values (all other drivers).</li> </ol>









### SQL to C data conversion in CLI

For a given SQL data type:

- the first column of the table lists the legal input values of the *fCType* argument in `SQLBindCol()` and `SQLGetData()`.
- the second column lists the outcomes of a test, often using the *cbValueMax* argument specified in `SQLBindCol()` or `SQLGetData()`, which the driver performs to determine if it can convert the data.
- the third and fourth columns list the values (for each outcome) of the *rgbValue* and *pcbValue* arguments specified in the `SQLBindCol()` or `SQLGetData()` after the driver has attempted to convert the data.
- the last column lists the `SQLSTATE` returned for each outcome by `SQLFetch()`, `SQLExtendedFetch()`, `SQLGetData()` or `SQLGetSubString()`.

The tables list the conversions defined by ODBC to be valid for a given SQL data type.

If the *fCType* argument in `SQLBindCol()` or `SQLGetData()` contains a value not shown in the table for a given SQL data type, `SQLFetch()`, or `SQLGetData()` returns the `SQLSTATE 07006` (Restricted data type attribute violation).

If the *fCType* argument contains a value shown in the table but which specifies a conversion not supported by the driver, `SQLFetch()`, or `SQLGetData()` returns `SQLSTATE HYC00` (Driver not capable).

Though it is not shown in the tables, the *pcbValue* argument contains `SQL_NULL_DATA` when the SQL data value is `NULL`. For an explanation of the use of *pcbValue* when multiple calls are made to retrieve data, see `SQLGetData()`.

When SQL data is converted to character C data, the character count returned in *pcbValue* does not include the null termination byte. If *rgbValue* is a null pointer, `SQLBindCol()` or `SQLGetData()` returns `SQLSTATE HY009` (Invalid argument value).

In the following tables:

#### **Length of data**

the total length of the data after it has been converted to the specified C data type (excluding the null termination byte if the data was converted to a string). This is true even if data is truncated before it is returned to the application.

#### **Significant digits**

the minus sign (if needed) and the digits to the left of the decimal point.

#### **Display size**

the total number of bytes needed to display data in the character format.

### **Converting character SQL data to C data**

The character SQL data types are:

- `SQL_CHAR`
- `SQL_VARCHAR`
- `SQL_LONGVARCHAR`
- `SQL_CLOB`

Table 180. Converting character SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Length of data < cbValueMax	Data	Length of data	00000
	Length of data >= cbValueMax	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	00000
	Length of data > cbValueMax	Truncated data	Length of data	01004
SQL_C_SHORT	Data converted without truncation <sup>a</sup>	Data	Size of the C data type	00000
SQL_C_LONG	Data converted with truncation, but without loss of significant digits <sup>a</sup>	Data	Size of the C data type	01004
SQL_C_FLOAT				
SQL_C_TINYINT				
SQL_C_BIT	Conversion of data would result in loss of significant digits <sup>a</sup>	Untouched	Size of the C data type	22003
SQL_C_UBIGINT				
SQL_C_SBIGINT				
SQL_C_NUMERIC <sup>c</sup>	Data is not a number <sup>a</sup>	Untouched	Size of the C data type	22005
SQL_C_TYPE_DATE	Data value is a valid date <sup>a</sup>	Data	6 <sup>b</sup>	00000
	Data value is not a valid date <sup>a</sup>	Untouched	6 <sup>b</sup>	22007
SQL_C_TYPE_TIME	Data value is a valid time <sup>a</sup>	Data	6 <sup>b</sup>	00000
	Data value is not a valid time <sup>a</sup>	Untouched	6 <sup>b</sup>	22007
SQL_C_TYPE_TIMESTAMP	Data value is a valid timestamp <sup>a</sup>	Data	16 <sup>b</sup>	00000
	Data value is not a valid timestamp <sup>a</sup>	Untouched	16 <sup>b</sup>	22007
SQL_C_TIMESTAMP_EXT	Data value is a valid timestamp <sup>a</sup>	Data	20	00000
	Data value is not a valid timestamp <sup>a</sup>	Untouched	20	22007

**Note:**

<sup>a</sup> The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

<sup>b</sup> This is the size of the corresponding C data type.

<sup>c</sup> SQL\_C\_NUMERIC is only supported on Windows platforms.

SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

## Converting graphic SQL data to C data

The graphic SQL data types are:

- SQL\_GRAPHIC
- SQL\_VARGRAPHIC
- SQL\_LONGVARGRAPHIC
- SQL\_DBCLOB

## SQL to C data conversion in CLI

Table 181. Converting GRAPHIC SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Number of double byte characters * 2 <= cbValueMax	Data	Length of data(octects)	00000
	Number of double byte characters * 2 > cbValueMax	Truncated data, to the nearest even byte that is less than cbValueMax.	Length of data(octects)	01004
SQL_C_DBCHAR	Number of double byte characters * 2 < cbValueMax	Data	Length of data(octects)	00000
	Number of double byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than cbValueMax.	Length of data(octects)	01004

**Note:** SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

When converting to floating point values, SQLSTATE 22003 will not be returned if non-significant digits of the resulting value are lost.

## Converting numeric SQL data to C data

The numeric SQL data types are:

- SQL\_DECIMAL
- SQL\_NUMERIC
- SQL\_SMALLINT
- SQL\_INTEGER
- SQL\_BIGINT
- SQL\_REAL
- SQL\_DECFLOAT
- SQL\_FLOAT
- SQL\_DOUBLE

Table 182. Converting numeric SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Display size < cbValueMax	Data	Length of data	00000
	Number of significant digits < cbValueMax	Truncated data	Length of data	01004
	Number of significant digits >= cbValueMax	Untouched	Length of data	22003
SQL_C_DBCHAR SQL_C_WCHAR	Display size * 2 < cbValueMax	Data	Length of data	00000
	Number of significant digits * 2 < cbValueMax	Truncated Data	Length of Data	01004
	Number of significant digits * 2 >= cbValueMax	Untouched	Length of Data	22003

Table 182. Converting numeric SQL data to C data (continued)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_SHORT	Data converted without truncation <sup>a</sup>	Data	Size of the C data type	00000
SQL_C_LONG	Data converted with truncation, but without loss of significant digits <sup>a</sup>	Truncated data	Size of the C data type	01004
SQL_C_FLOAT				
SQL_C_DOUBLE				
SQL_C_TINYINT	Conversion of data would result in loss of significant digits <sup>a</sup>	Untouched	Size of the C data type	22003
SQL_C_BIT				
SQL_C_UBIGINT				
SQL_C_SBIGINT				
SQL_C_NUMERIC <sup>b</sup>				

**Note:**

<sup>a</sup> The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

<sup>b</sup> SQL\_C\_NUMERIC is only supported on Windows platforms.

SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

## Converting binary SQL data to C data

The binary SQL data types are:

- SQL\_BINARY
- SQL\_VARBINARY
- SQL\_LONGVARBINARY
- SQL\_BLOB

Table 183. Converting binary SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	(Length of data) < cbValueMax	Data	Length of data	N/A
	(Length of data) >= cbValueMax	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	N/A
	Length of data > cbValueMax	Truncated data	Length of data	01004

**Note:** Starting in DB2 Version 9.7 Fix Pack 6, SQL\_BINARY and SQL\_VARBINARY data types are supported for DB2 for i Version 6 Release 1 servers or later releases.

## Converting XML SQL data to C data

The XML SQL data type is:

SQL\_XML

Table 184. Converting XML SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Length of data < cbValueMax	Data	Length of data	00000
	Length of data >= cbValueMax	Truncated data	Length of data	01004

## SQL to C data conversion in CLI

Table 184. Converting XML SQL data to C data (continued)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	00000
	Length of data > cbValueMax	Truncated data	Length of data	01004
SQL_C_BINARYXML	Length of data <= cbValueMax	Data	Length of data	00000
	Length of data > cbValueMax	Truncated data	Length of data	01004
SQL_C_DBCHAR	Number of double-byte characters * 2 < cbValueMax	Data	Length of data	00000
	Number of double-byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than cbValueMax	Length of data	01004
SQL_C_WCHAR	Number of double-byte characters * 2 < cbValueMax	Data	Length of data	00000
	Number of double-byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than cbValueMax	Length of data	01004

**Note:**

1. SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.
2. Length of data includes any XML declaration in the target encoding.
3. The SQL\_XML data type is not supported for use with an Informix data server.

## Converting date SQL data to C data

The date SQL data type is:

- SQL\_TYPE\_DATE

Table 185. Converting date SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 11	Data	10	00000
	cbValueMax < 11	Untouched	10	22003
SQL_C_TYPE_DATE	None <sup>a</sup>	Data	6 <sup>b</sup>	00000
SQL_C_TYPE_TIMESTAMP	None <sup>a</sup>	Data <sup>c</sup>	16 <sup>b</sup>	00000
SQL_C_TIMESTAMP_EXT	None <sup>a</sup>	Data <sup>c</sup>	20	00000

**Note:**

- <sup>a</sup> The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.
- <sup>b</sup> This is the size of the corresponding C data type.
- <sup>c</sup> The time fields of the `TIMESTAMP_STRUCT` or `TIMESTAMP_STRUCT_EXT` structure are set to zero.

SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.



When the date SQL data type is converted to the character C data type, the resulting string is in the "yyyy-mm-dd" format.

## Converting Time SQL Data to C Data

The time SQL data type is:

- SQL\_TYPE\_TIME

Table 186. Converting time SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 9	Data	8	00000
	cbValueMax < 9	Untouched	8	22003
SQL_C_TYPE_TIME	None <sup>a</sup>	Data	6 <sup>b</sup>	00000
SQL_C_TYPE_TIMESTAMP	None <sup>a</sup>	Data <sup>c</sup>	16 <sup>b</sup>	00000
SQL_C_TIMESTAMP_EXT	None <sup>a</sup>	Data <sup>c</sup>	20	00000

**Note:**

<sup>a</sup> The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

<sup>b</sup> This is the size of the corresponding C data type.

<sup>c</sup> The date fields of the `TIMESTAMP_STRUCT` or `TIMESTAMP_STRUCT_EXT` structure are set to the current system date of the machine that the application is running, and the time fraction is set to zero.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

When the time SQL data type is converted to the character C data type, the resulting string is in the "hh:mm:ss" format.

## Converting timestamp SQL data to C data

The timestamp SQL data type is:

- SQL\_TYPE\_TIMESTAMP

Table 187. Converting timestamp SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Display size < cbValueMax	Data	Length of data	00000
	19 <= cbValueMax <= Display size	Truncated Data <sup>b</sup>	Length of data	01004
	cbValueMax < 19	Untouched	Length of data	22003
SQL_C_TYPE_DATE	None <sup>a</sup>	Truncated data <sup>c</sup>	6 <sup>e</sup>	01S07
SQL_C_TYPE_TIME	None <sup>a</sup>	Truncated data <sup>d</sup>	6 <sup>e</sup>	01S07
SQL_C_TYPE_TIMESTAMP	None <sup>a</sup>	Data	16 <sup>e</sup>	00000
SQL_C_TIMESTAMP_EXT	None <sup>a</sup>	Data	20	00000

## SQL to C data conversion in CLI

Table 187. Converting timestamp SQL data to C data (continued)

fCType	Test	rgbValue	pcbValue	SQLSTATE
<b>Note:</b>				
<sup>a</sup>	The value of <i>cbValueMax</i> is ignored for this conversion. The driver assumes that the size of <i>rgbValue</i> is the size of the C data type.			
<sup>b</sup>	The fractional seconds of the timestamp are truncated.			
<sup>c</sup>	The time portion of the timestamp is deleted.			
<sup>d</sup>	The date portion of the timestamp is deleted.			
<sup>e</sup>	This is the size of the corresponding C data type.			
SQLSTATE 00000 is not returned by <code>SQLGetDiagRec()</code> , rather it is indicated when the function returns <code>SQL_SUCCESS</code> .				

When the timestamp SQL data type is converted to the character C data type, the resulting string is in the "yyyy-mm-dd hh:mm:ss.fffffffffff" format where fractional second digits range from 0 to 12 (regardless of the precision of the timestamp SQL data type). If an application requires the ISO format, set the CLI/ODBC configuration keyword `PATCH2=33`.

### Converting timestamp(p) with timezone SQL data to C data

The timestamp SQL data type is:

- `SQL_TYPE_TIMESTAMP_WITH_TIMEZONE`

Table 188. Converting timestamp(p) with timezone SQL data to C data

fCType	SQL Type	Test/Result	SQLSTATE
<code>SQL_C_TYPE_TIMESTAMP_EXT_TZ</code>	<code>SQL_CHAR</code>	Data value is a valid timestamp with timezone	N/A
		Data value is not a valid timestamp with timezone	22007
<code>SQL_C_TYPE_DATE</code>	<code>SQL_TYPE_TIMESTAMP_WITH_TIMEZONE</code>	Truncated data	01S07
<code>SQL_C_TYPE_TIME</code>	<code>SQL_TYPE_TIMESTAMP_WITH_TIMEZONE</code>	Truncated data	01S07
<code>SQL_C_TYPE_TIMESTAMP</code>	<code>SQL_TYPE_TIMESTAMP_WITH_TIMEZONE</code>	Truncated data	01S07

### SQL to C data conversion examples

Table 189. SQL to C data conversion examples

SQL data type	SQL data value	C data type	cbValue max	rgbValue	SQL STATE
<code>SQL_CHAR</code>	abcdef	<code>SQL_C_CHAR</code>	7	abcdef\0 <sup>a</sup>	00000
<code>SQL_CHAR</code>	abcdef	<code>SQL_C_CHAR</code>	6	abcde\0 <sup>a</sup>	01004
<code>SQL_DECIMAL</code>	1234.56	<code>SQL_C_CHAR</code>	8	1234.56\0 <sup>a</sup>	00000
<code>SQL_DECIMAL</code>	1234.56	<code>SQL_C_CHAR</code>	5	1234\0 <sup>a</sup>	01004
<code>SQL_DECIMAL</code>	1234.56	<code>SQL_C_CHAR</code>	4	---	22003
<code>SQL_DECIMAL</code>	1234.56	<code>SQL_C_FLOAT</code>	ignored	1234.56	00000

Table 189. SQL to C data conversion examples (continued)

SQL data type	SQL data value	C data type	cbValue max	rgbValue	SQL STATE
SQL_DECIMAL	1234.56	SQL_C_SHORT	ignored	1234	01004
SQL_TYPE_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31\0 <sup>a</sup>	00000
SQL_TYPE_DATE	1992-12-31	SQL_C_CHAR	10	---	22003
SQL_TYPE_DATE	1992-12-31	SQL_C_TYPE_TIMESTAMP	ignored	1992,12,31, 0,0,0,0 <sup>b</sup>	00000
SQL_TYPE_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12\0 <sup>a</sup>	00000
SQL_TYPE_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1\0 <sup>a</sup>	01004
SQL_TYPE_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	---	22003

**Note:**

<sup>a</sup> "\0" represents a null termination character.

<sup>b</sup> The numbers in this list are the numbers stored in the fields of the `TIMESTAMP_STRUCT` or `TIMESTAMP_STRUCT_EXT` structure.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

## C to SQL data conversion in CLI

For a given C data type:

- the first column of the table lists the legal input values of the *fsqlType* argument in `SQLBindParameter()` or `SQLSetParam()`.
- the second column lists the outcomes of a test, often using the length of the parameter data as specified in the *pcbValue* argument in `SQLBindParameter()` or `SQLSetParam()`, which the driver performs to determine if it can convert the data.
- the third column lists the SQLSTATE returned for each outcome by `SQLExecDirect()` or `SQLExecute()`.

The tables list the conversions defined by ODBC to be valid for a given SQL data type.

If the *fsqlType* argument in `SQLBindParameter()` or `SQLSetParam()` contains a value not shown in the table for a given C data type, SQLSTATE 07006 is returned (Restricted data type attribute violation).

If the *fsqlType* argument contains a value shown in the table but which specifies a conversion not supported by the driver, `SQLBindParameter()` or `SQLSetParam()` returns SQLSTATE HYC00 (Driver not capable).

If the *rgbValue* and *pcbValue* arguments specified in `SQLBindParameter()` or `SQLSetParam()` are both null pointers, that function returns SQLSTATE HY009 (Invalid argument value).

**Note:**

- The `SQL_XML` data type is not supported for use with an Informix data server.

## C to SQL data conversion in CLI

- Starting in DB2 Version 9.7 Fix Pack 6, SQL\_BINARY and SQL\_VARBINARY data types are supported for DB2 for i Version 6 Release 1 servers or later releases.

### Length of data

the total length of the data after it has been converted to the specified SQL data type (excluding the null termination byte if the data was converted to a string). This is true even if data is truncated before it is sent to the data source.

### Column length

the maximum number of bytes returned to the application when data is transferred to its default C data type. For character data, the length does not include the null termination byte.

### Display size

the maximum number of bytes needed to display data in character form.

### Significant digits

the minus sign (if needed) and the digits to the left of the decimal point.

## Converting character C data to SQL data

The character C data type is:

- SQL\_C\_CHAR

*Table 190. Converting character C data to SQL data*

fsQLType	Test	SQLSTATE	
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data <= Column length	N/A	
	Length of data > Column length	22001	
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	Data converted without truncation	N/A	
	Data converted with truncation, but without loss of significant digits	22001	
	Conversion of data would result in loss of significant digits	22003	
	Data value is not a numeric value	22005	
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	(Length of data) < Column length	N/A	
	(Length of data) >= Column length	22001	
	Data value is not a hexadecimal value	22005	
SQL_TYPE_DATE	Data value is a valid date	N/A	
	Data value is not a valid date	22007	
	Data value is a valid timestamp	22008	
	Data value is a valid timestamp and the connection attribute SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN is set to 1	01S07 (Fractional truncation warning)	
SQL_TYPE_TIME	Data value is a valid time	N/A	
	Data value is not a valid time	22007	
	Data value is a valid timestamp	22008	
	Data value is a valid timestamp and the connection attribute SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN is set to 1	01S07 (Fractional truncation warning)	
SQL_TYPE_TIMESTAMP	Data value is a valid timestamp	N/A	
	Data value is not a valid timestamp	22007	
	Data value is a valid date	N/A	
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLCLOB	Length of data / 2 <= Column length	N/A	
	Length of data / 2 < Column length	22001	
	SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

## Converting numeric C data to SQL data

The numeric C data types are:

- SQL\_C\_SHORT
- SQL\_C\_LONG
- SQL\_C\_FLOAT
- SQL\_C\_DOUBLE
- SQL\_C\_TINYINT
- SQL\_C\_SBIGINT
- SQL\_C\_BIT

*Table 191. Converting numeric C data to SQL data*

isQLType	Test	SQLSTATE
	Data converted without truncation	N/A
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	Data converted with truncation, but without loss of significant digits	22001
	Conversion of data would result in loss of significant digits	22003
	Data converted without truncation.	N/A
SQL_CHAR SQL_VARCHAR	Conversion of data would result in loss of significant digits.	22003

Note: When converting to floating point values, SQLSTATE 22003 will not be returned if non-significant digits of the resulting value are lost.

## Converting binary C data to SQL data

The binary C data type is:

- SQL\_C\_BINARY

*Table 192. Converting binary C data to SQL data*

isQLType	Test	SQLSTATE
	Length of data <= Column length	N/A
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data > Column length	22001
	Length of data <= Column length	N/A
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	Length of data > Column length	22001
SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

## Converting binary XML C data to SQL data

The binary XML C data type is:

- SQL\_C\_BINARYXML

*Table 193. Converting binary XML C data to SQL data*

isQLType	Test	SQLSTATE
SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

## Converting DBCHAR C data to SQL data

The double byte C data type is:

- SQL\_C\_DBCHAR

Table 194. Converting DBCHAR C data to SQL data

fsQLType	Test	SQLSTATE	
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data <= Column length x 2	N/A	
	Length of data > Column length x 2	22001	
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DECFLOAT SQL_DOUBLE	Length of data <= Column length x 2	N/A	
	Length of data > Column length x 2	22001	
	Data is non-Unicode	07006	
	Length of data <= Column length x 2	N/A	
	Length of data > Column length x 2	22001	
	SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB		
	SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

## Converting date C data to SQL data

The date C data type is:

- SQL\_C\_DATE

Table 195. Converting date C data to SQL data

fsQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= 10	N/A
	Column length < 10	22001
SQL_TYPE_DATE	Data value is a valid date	N/A
	Data value is not a valid date	22007
SQL_TYPE_TIMESTAMP <sup>a</sup>	Data value is a valid date	N/A
	Data value is not a valid date	22007

Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

Note: a, the time component of TIMESTAMP is set to zero.

## Converting time C data to SQL data

The time C data type is:

- SQL\_C\_TIME

Table 196. Converting time C data to SQL data

fsQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= 8	N/A
	Column length < 8	22001
SQL_TYPE_TIME	Data value is a valid time	N/A
	Data value is not a valid time	22007
SQL_TYPE_TIMESTAMP <sup>a</sup>	Data value is a valid time	N/A
	Data value is not a valid time	22007

Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

Note: a The date component of TIMESTAMP is set to the system date of the machine at which the application is running.

## Converting timestamp C data to SQL data

The timestamp C data type is:

- SQL\_C\_TIMESTAMP

*Table 197. Converting timestamp C data to SQL data*

ISQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= Display size	N/A
	26 <= Column length < Display size <sup>a</sup>	N/A
	Column length < 26	22001
SQL_TYPE_DATE	Time fields are zero	N/A
	Time fields are non-zero	22008
	Data value does not contain a valid date <sup>b</sup>	22007
	Time fields are non-zero and the connection attribute SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN is set to 1	01S07 (Fractional truncation warning)
SQL_TYPE_TIME	Fractional seconds fields are zero	N/A
	Fractional seconds fields are non-zero	22008
	Data value does not contain a valid time	22007
SQL_TYPE_TIMESTAMP	Time fields are non-zero and the connection attribute SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN is set to 1	01S07 (Fractional truncation warning)
	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007

**Note:**

<sup>a</sup> The fractional seconds of the timestamp are truncated.

<sup>b</sup> The timestamp\_struct must reset the hour, minute, second, and fraction to 0, otherwise SQLSTATE 22008 will be returned.

SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

## Converting variable timestamp C data to SQL data

The timestamp C data type is:

- SQL\_C\_TIMESTAMP\_EXT

*Table 198. Converting variable timestamp C data to SQL data*

ISQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= Display size	N/A
	26 <= Column length < Display size <sup>a</sup>	N/A
	Column length < 26	22001
SQL_TYPE_DATE	Fractional seconds fields > 12	22007
	Time fields are zero	N/A
	Time fields are non-zero	22008
SQL_TYPE_TIME	Data value does not contain a valid date <sup>b</sup>	22007
	Fractional seconds fields are zero	N/A
	Fractional seconds fields are non-zero	22008
SQL_TYPE_TIMESTAMP	Data value does not contain a valid time	22007
	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007
	Precision specified by TIMESTAMP(p) <= fractional seconds fields <= 12 <sup>a</sup>	N/A

**Note:**

<sup>a</sup> The fractional seconds of the timestamp are truncated.

<sup>b</sup> The timestamp\_struct must reset the hour, minute, second, and fraction to 0, otherwise SQLSTATE 22008 will be returned.

SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

## Converting timestamp(p) with timezone C data to SQL data

The timestamp with timezone C data type is:

- SQL\_C\_TIMESTAMP\_EXT\_TZ

*Table 199. Converting timestamp with timezone C data to SQL data*

ISQLType	Test	SQLSTATE
SQL_CHAR /SQL_VARCHAR SQL_WCHAR/SQL_WVARCHAR SQL_LONGVARCHAR	Column length >= Display size	N/A
	Column length < 27+p	22001
SQL_TYPE_DATE	Time fields are zero	N/A
	Time fields are non-zero	22008
	Data value does not contain a valid date	22007
	Time zone fields are non-zero	22008

## C to SQL data conversion in CLI

*Table 199. Converting timestamp with timezone C data to SQL data (continued)*

fSqlType	Test	SQLSTATE
SQL_TYPE_TIME	Fractional seconds fields are zero	N/A
	Fractional seconds fields are non-zero	22008
	Data value does not contain a valid time	22007
	Time zone fields are non-zero	22008
SQL_TYPE_TIMESTAMP	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007
	Time zone fields are non-zero	22008
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	Data value is a valid timestamp	N/A
	Data value is a valid timestamp with time zone	22007

## C to SQL data conversion examples

*Table 200. C to SQL data conversion examples*

C data type	C data value	SQL data type	Column length	SQL data value	SQL STATE
SQL_C_CHAR	abcdef\0	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef\0	SQL_CHAR	5	abcde	22001
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	6	1234.56	N/A
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	5	1234.5	22001
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	3	---	22003
SQL_C_CHAR	4.46.32	SQL_TYPE_TIME	6	4.46.32	N/A
SQL_C_CHAR	4-46-32	SQL_TYPE_TIME	6		22007
				not applicable	
SQL_C_DOUBLE	123.45	SQL_CHAR	22		N/A
				1.23450000 000000e+02	
SQL_C_FLOAT	1234.56	SQL_FLOAT		1234.56	N/A
			not applicable		
SQL_C_FLOAT	1234.56	SQL_INTEGER		1234	22001
			not applicable		
		SQL_TYPE_DATE	6	1992-12-31	01004
SQL_C_TIMESTAMP	1992-12-31 23:45:55. 123456				
SQL_C_TIMESTAMP_EXT	2009-06-06 23:45:55. 123456789876	SQL_TYPE_DATE	6	2009-06-06	01004

Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL\_SUCCESS.

## Data type attributes

### Data type precision (CLI) table

The precision of a numeric column or parameter refers to the maximum number of digits used by the data type of the column or parameter. The precision of a non-numeric column or parameter generally refers to the maximum or the defined number of characters of the column or parameter. The following table defines the precision for each SQL data type.

*Table 201. Precision*

fSqlType	Precision
SQL_CHAR SQL_VARCHAR SQL_CLOB	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR	The maximum length of the column or parameter. <sup>a</sup>



Table 201. Precision (continued)

fSqlType	Precision
SQL_DECIMAL SQL_DECFLOAT SQL_NUMERIC	The defined maximum number of digits. For example, the precision of a column defined as NUMERIC(10,3) is 10 and the precision of a column defined as DECFLOAT(34) is 34.
SQL_SMALLINT <sup>b</sup>	5
SQL_BIGINT	19
SQL_INTEGER <sup>b</sup>	10
SQL_FLOAT <sup>b</sup>	15
SQL_REAL <sup>b</sup>	7
SQL_DOUBLE <sup>b</sup>	15
SQL_BINARY SQL_VARBINARY SQL_BLOB	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) FOR BIT DATA, is 10.
SQL_LONGVARBINARY	The maximum length of the column or parameter.
SQL_TYPE_DATE <sup>b</sup>	10 (the number of characters in the yyyy-mm-dd format).
SQL_TYPE_TIME <sup>b</sup>	8 (the number of characters in the hh:mm:ss format).
SQL_TYPE_TIMESTAMP	The number of characters in the "yyyy-mm-dd hh:mm:ss[.fffffffffff]" format used by the TIMESTAMP data type. For example, if a timestamp does not use seconds or fractional seconds, the precision is 16 (the number of characters in the "yyyy-mm-dd hh:mm" format). If a timestamp uses thousandths of a second, the precision is 23 (the number of characters in the "yyyy-mm-dd hh:mm:ss.fff" format).
SQL_TYPE_TIMESTAMP_ WITH_TIMEZONE	The number of characters in the "yyyy-mm-dd hh:mm:ss[.fffffffffff]" format used by the TIMESTAMP_WITH_TIMEZONE data type. The valid range for the time zone is -12:59~ +14:00. If the timezone_hour is negative, the timezone_minute must be negative or zero. If the timezone_hour is positive, the timezone_minute must be positive or zero. If the timezone_hour is zero, the timezone_minute can have any value in the range -59 through +59.
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	The defined length of the column or parameter. For example, the precision of a column defined as GRAPHIC(10) is 10.
SQL_LONGVARGRAPHIC	The maximum length of the column or parameter.
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	The defined length of the column or parameter. For example, the precision of a column defined as WCHAR(10) is 10.
SQL_XML	0, unless the XML value is an argument to external routines. For external routines, the precision is the defined length, n, of an XML AS CLOB(n) argument.

## Data type precision (CLI) table

Table 201. Precision (continued)

fSqlType	Precision
<b>Note:</b>	
<sup>a</sup>	When defining the precision of a parameter of this data type with SQLBindParameter() or SQLSetParam(), <i>cbParamDef</i> should be set to the total length of the data, not the precision as defined in this table.
<sup>b</sup>	The <i>cbColDef</i> argument of SQLBindParameter() is ignored for this data type.

## Data type scale (CLI) table

The scale of a numeric column or parameter refers to the maximum number of digits to the right of the decimal point. Note that, for approximate floating point number columns or parameters, the scale is undefined, since the number of digits to the right of the decimal place is not fixed. The following table defines the scale for each SQL data type.

Table 202. Scale

fSqlType	Scale
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Not applicable.
SQL_DECIMAL SQL_NUMERIC	The defined number of digits to the right of the decimal place. For example, the scale of a column defined as NUMERIC(10, 3) is 3.
SQL_SMALLINT SQL_INTEGER SQL_BIGINT	0
SQL_REAL SQL_FLOAT SQL_DECFLOAT SQL_DOUBLE	Not applicable.
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	Not applicable.
SQL_TYPE_DATE SQL_TYPE_TIME	Not applicable.
SQL_TYPE_TIMESTAMP	The number of digits to the right of the decimal point in the "yyyy-mm-dd hh:mm:ss[.fffffffffff]" format. For example, if the TIMESTAMP data type uses the "yyyy-mm-dd hh:mm:ss.fff" format, the scale is 3.

Table 202. Scale (continued)

fSqlType	Scale
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLOB	Not applicable.
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	Not applicable.
SQL_XML	Not applicable.

## Data type length (CLI) table

The length of a column is the maximum number of *bytes* returned to the application when data is transferred to its default C data type. For character data, the length does not include the null termination byte. Note that the length of a column might be different than the number of bytes required to store the data on the data source.

The following table defines the length for each SQL data type.

Table 203. Length

fSqlType	Length
SQL_CHAR SQL_VARCHAR SQL_CLOB	The defined length of the column. For example, the length of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR	The maximum length of the column.
SQL_DECIMAL SQL_NUMERIC	The maximum number of digits plus two. Since these data types are returned as character strings, characters are needed for the digits, a sign, and a decimal point. For example, the length of a column defined as NUMERIC(10,3) is 12.
SQL_DECFLOAT	If the column is defined as DECFLOAT(16) then the length is 8. If the column is defined as DECFLOAT(34) then the length is 16.
SQL_SMALLINT	2 (two bytes).
SQL_INTEGER	4 (four bytes).
SQL_BIGINT	8 (eight bytes).
SQL_REAL	4 (four bytes).

## Data type length (CLI) table

Table 203. Length (continued)

fSqlType	Length
SQL_FLOAT	8 (eight bytes).
SQL_DOUBLE	8 (eight bytes).
SQL_BINARY SQL_VARBINARY SQL_BLOB	The defined length of the column. For example, the length of a column defined as CHAR(10) FOR BIT DATA is 10.
SQL_LONGVARBINARY	The maximum length of the column.
SQL_TYPE_DATE SQL_TYPE_TIME	6 (the size of the DATE_STRUCT or TIME_STRUCT structure).
SQL_TYPE_TIMESTAMP	16 (the size of the TIMESTAMP_STRUCT structure).
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	The defined length of the column times 2. For example, the length of a column defined as GRAPHIC(10) is 20.
SQL_LONGVARGRAPHIC	The maximum length of the column times 2.
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	The defined length of the column times 2. For example, the length of a column defined as WCHAR(10) is 20.
SQL_XML	0 (stored XML documents are limited to 2GB in size however)

## Data type display (CLI) table

The display size of a column is the maximum number of *bytes* needed to display data in character form. The following table defines the display size for each SQL data type.

Table 204. Display size

fSqlType	Display size
SQL_CHAR SQL_VARCHAR SQL_CLOB	The defined length of the column. For example, the display size of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR	The maximum length of the column.
SQL_DECIMAL SQL_NUMERIC	The precision of the column plus two (a sign, precision digits, and a decimal point). For example, the display size of a column defined as NUMERIC(10,3) is 12.

Table 204. Display size (continued)

fSqlType	Display size
SQL_DECFLOAT	If the column is defined as DECFLOAT(16) then the display length is 24. If the column is defined as DECFLOAT(34) then the display length is 42.
SQL_SMALLINT	6 (a sign and 5 digits).
SQL_INTEGER	11 (a sign and 10 digits).
SQL_BIGINT	20 (a sign and 19 digits).
SQL_REAL	13 (a sign, 7 digits, a decimal point, the letter E, a sign, and 2 digits).
SQL_FLOAT SQL_DOUBLE	22 (a sign, 15 digits, a decimal point, the letter E, a sign, and 3 digits).
SQL_BINARY SQL_VARBINARY SQL_BLOB	The defined maximum length of the column times 2 (each binary byte is represented by a 2 digit hexadecimal number). For example, the display size of a column defined as CHAR(10) FOR BIT DATA is 20.
SQL_LONGVARBINARY	The maximum length of the column times 2.
SQL_TYPE_DATE	10 (a date in the format yyyy-mm-dd).
SQL_TYPE_TIME	8 (a time in the format hh:mm:ss).
SQL_TYPE_TIMESTAMP	19 (if the scale of the timestamp is 0) or 20 plus the scale of the timestamp (if the scale is greater than 0). This is the number of characters in the "yyyy-mm-dd hh:mm:ss[.fffffffffff]" format. For example, the display size of a column storing thousandths of a second is 23 (the number of characters in "yyyy-mm-dd hh:mm:ss.fff").
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	Twice the defined length of the column or parameter. For example, the display size of a column defined as GRAPHIC(10) is 20.
SQL_LONGVARGRAPHIC	The maximum length of the column or parameter.
SQL_XML	0

## Data type display (CLI) table

---

## Appendix A. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

**Note:** The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at [ibm.com](http://ibm.com).

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at [ibm.com](http://ibm.com). Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

### Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to [db2docs@ca.ibm.com](mailto:db2docs@ca.ibm.com). The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

---

## DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at [www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss](http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss). English Version 9.7 manuals in PDF format can be downloaded from [www.ibm.com/support/docview.wss?uid=swg27015148](http://www.ibm.com/support/docview.wss?uid=swg27015148) and translated DB2 manuals in PDF format can be downloaded from [www.ibm.com/support/docview.wss?uid=swg27015149](http://www.ibm.com/support/docview.wss?uid=swg27015149).

## DB2 technical library in hardcopy or PDF format

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

**Note:** The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

*Table 205. DB2 technical information*

<b>Name</b>	<b>Form Number</b>	<b>Available in print</b>	<b>Last updated</b>
<i>Administrative API Reference</i>	SC27-2435-03	Yes	July, 2012
<i>Administrative Routines and Views</i>	SC27-2436-03	No	July, 2012
<i>Call Level Interface Guide and Reference, Volume 1</i>	SC27-2437-03	Yes	July, 2012
<i>Call Level Interface Guide and Reference, Volume 2</i>	SC27-2438-03	Yes	July, 2012
<i>Command Reference</i>	SC27-2439-03	Yes	July, 2012
<i>Data Movement Utilities Guide and Reference</i>	SC27-2440-01	Yes	July, 2012
<i>Data Recovery and High Availability Guide and Reference</i>	SC27-2441-03	Yes	July, 2012
<i>Database Administration Concepts and Configuration Reference</i>	SC27-2442-03	Yes	July, 2012
<i>Database Monitoring Guide and Reference</i>	SC27-2458-03	Yes	July, 2012
<i>Database Security Guide</i>	SC27-2443-02	Yes	July, 2012
<i>DB2 Text Search Guide</i>	SC27-2459-03	Yes	July, 2012
<i>Developing ADO.NET and OLE DB Applications</i>	SC27-2444-02	Yes	July, 2012
<i>Developing Embedded SQL Applications</i>	SC27-2445-02	Yes	July, 2012
<i>Developing Java Applications</i>	SC27-2446-03	Yes	July, 2012
<i>Developing Perl, PHP, Python, and Ruby on Rails Applications</i>	SC27-2447-02	No	July, 2012
<i>Developing User-defined Routines (SQL and External)</i>	SC27-2448-02	Yes	July, 2012
<i>Getting Started with Database Application Development</i>	GI11-9410-02	Yes	July, 2012



Table 205. DB2 technical information (continued)

<b>Name</b>	<b>Form Number</b>	<b>Available in print</b>	<b>Last updated</b>
<i>Getting Started with DB2 Installation and Administration on Linux and Windows</i>	GI11-9411-00	Yes	August, 2009
<i>Globalization Guide</i>	SC27-2449-00	Yes	August, 2009
<i>Installing DB2 Servers</i>	GC27-2455-03	Yes	July, 2012
<i>Installing IBM Data Server Clients</i>	GC27-2454-02	No	July, 2012
<i>Message Reference Volume 1</i>	SC27-2450-01	No	August, 2009
<i>Message Reference Volume 2</i>	SC27-2451-01	No	August, 2009
<i>Net Search Extender Administration and User's Guide</i>	SC27-2469-02	No	September, 2010
<i>Partitioning and Clustering Guide</i>	SC27-2453-02	Yes	July, 2012
<i>pureXML Guide</i>	SC27-2465-02	Yes	July, 2012
<i>Query Patroller Administration and User's Guide</i>	SC27-2467-00	No	August, 2009
<i>Spatial Extender and Geodetic Data Management Feature User's Guide and Reference</i>	SC27-2468-02	No	July, 2012
<i>SQL Procedural Languages: Application Enablement and Support</i>	SC27-2470-03	Yes	July, 2012
<i>SQL Reference, Volume 1</i>	SC27-2456-03	Yes	July, 2012
<i>SQL Reference, Volume 2</i>	SC27-2457-03	Yes	July, 2012
<i>Troubleshooting and Tuning Database Performance</i>	SC27-2461-03	Yes	July, 2012
<i>Upgrading to DB2 Version 9.7</i>	SC27-2452-03	Yes	July, 2012
<i>Visual Explain Tutorial</i>	SC27-2462-00	No	August, 2009
<i>What's New for DB2 Version 9.7</i>	SC27-2463-03	Yes	July, 2012
<i>Workload Manager Guide and Reference</i>	SC27-2464-03	Yes	July, 2012
<i>XQuery Reference</i>	SC27-2466-01	No	November, 2009

## DB2 technical library in hardcopy or PDF format

Table 206. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
<i>Installing and Configuring DB2 Connect Personal Edition</i>	SC27-2432-03	Yes	July, 2012
<i>Installing and Configuring DB2 Connect Servers</i>	SC27-2433-03	Yes	July, 2012
<i>DB2 Connect User's Guide</i>	SC27-2434-02	Yes	July, 2012

Table 207. Information Integration technical information

Name	Form Number	Available in print	Last updated
<i>Information Integration: Administration Guide for Federated Systems</i>	SC19-1020-02	Yes	August, 2009
<i>Information Integration: ASNCLP Program Reference for Replication and Event Publishing</i>	SC19-1018-04	Yes	August, 2009
<i>Information Integration: Configuration Guide for Federated Data Sources</i>	SC19-1034-02	No	August, 2009
<i>Information Integration: SQL Replication Guide and Reference</i>	SC19-1030-02	Yes	August, 2009
<i>Information Integration: Introduction to Replication and Event Publishing</i>	GC19-1028-02	Yes	August, 2009

---

## Ordering printed DB2 books

### About this task

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation DVD* are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

**Note:** The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7>.

To order printed DB2 books:

### Procedure

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at <http://www.ibm.com/shop/publications/order>. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following websites:
    - The IBM directory of world wide contacts at [www.ibm.com/planetwide](http://www.ibm.com/planetwide)
    - The IBM Publications website at <http://www.ibm.com/shop/publications/order>. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  2. When you call, specify that you want to order a DB2 publication.
  3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see “DB2 technical library in hardcopy or PDF format” on page 539.

---

## Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

### Procedure

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, ? 08003 displays help for the 08003 SQL state, and ? 08 displays help for the 08 class code.

---

## Accessing different versions of the DB2 Information Center

### About this task

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/>.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/>.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/>.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>.

## Accessing different versions of the DB2 Information Center

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at:  
<http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

---

## Displaying topics in your preferred language in the DB2 Information Center

### About this task

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

### Procedure

- To display topics in your preferred language in the Internet Explorer browser:
  1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
    - To add a new language to the list, click the **Add...** button.

**Note:** Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
    - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Refresh the page to display the DB2 Information Center in your preferred language.
- To display topics in your preferred language in a Firefox or Mozilla browser:
  1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
    - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
    - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Refresh the page to display the DB2 Information Center in your preferred language.

### Results

On some browser and operating system combinations, you must also change the regional settings of your operating system to the locale and language of your choice.

---

## Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

## Updating the DB2 Information Center installed on your computer or intranet server

### Before you begin

A DB2 Version 9.7 Information Center must already be installed. For details, see the “Installing the DB2 Information Center using the DB2 Setup wizard” topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

### About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates - updates existing Information Center features and languages. An additional benefit of automatic updates is that the Information Center is unavailable for a minimal period of time during the update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates - should be used when you want to add features or languages during the update process. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process.

This topic details the process for automatic updates. For manual update instructions, see the “Manually updating the DB2 Information Center installed on your computer or intranet server” topic.

### Procedure

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
  - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9.7` directory.
  - b. Navigate from the installation directory to the `doc/bin` directory.
  - c. Run the `update-ic` script:  
`update-ic`
2. On Windows operating systems,
  - a. Open a command window.
  - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `<Program Files>\IBM\DB2 Information Center\Version 9.7` directory, where `<Program Files>` represents the location of the Program Files directory.
  - c. Navigate from the installation directory to the `doc\bin` directory.
  - d. Run the `update-ic.bat` file:  
`update-ic.bat`

### Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center

## Updating the DB2 Information Center installed on your computer or intranet server

updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

---

## Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

### About this task

Updating your locally-installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

**Note:** If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.

If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

**Note:** On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

### Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
  - On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Stop**.
  - On Linux, enter the following command:  

```
/etc/init.d/db2icdv97 stop
```
2. Start the Information Center in stand-alone mode.
  - On Windows:
    - a. Open a command window.
    - b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the

## Manually updating the DB2 Information Center installed on your computer or intranet server

*Program\_Files\IBM\DB2 Information Center\Version 9.7* directory, where *Program\_Files* represents the location of the Program Files directory.

c. Navigate from the installation directory to the `doc\bin` directory.

d. Run the `help_start.bat` file:

```
help_start.bat
```

• On Linux:

a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the `/opt/ibm/db2ic/V9.7` directory.

b. Navigate from the installation directory to the `doc/bin` directory.

c. Run the `help_start` script:

```
help_start
```

The systems default Web browser opens to display the stand-alone Information Center.

3. Click the **Update** button (🔄). (JavaScript must be enabled in your browser.)

On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.

4. To initiate the installation process, check the selections you want to install, then click **Install Updates**.

5. After the installation process has completed, click **Finish**.

6. Stop the stand-alone Information Center:

• On Windows, navigate to the installation directory's `doc\bin` directory, and run the `help_end.bat` file:

```
help_end.bat
```

**Note:** The `help_end` batch file contains the commands required to safely stop the processes that were started with the `help_start` batch file. Do not use `Ctrl-C` or any other method to stop `help_start.bat`.

• On Linux, navigate to the installation directory's `doc/bin` directory, and run the `help_end` script:

```
help_end
```

**Note:** The `help_end` script contains the commands required to safely stop the processes that were started with the `help_start` script. Do not use any other method to stop the `help_start` script.

7. Restart the *DB2 Information Center*.

• On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Start**.

• On Linux, enter the following command:

```
/etc/init.d/db2icdv97 start
```

### Results

The updated *DB2 Information Center* displays the new and updated topics.

---

## DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

### Before you begin

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2help/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

### DB2 tutorials

To view the tutorial, click the title.

#### **“pureXML®” in *pureXML Guide***

Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

#### **“Visual Explain” in *Visual Explain Tutorial***

Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

---

## DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you with using DB2 database products.

### DB2 documentation

Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*. The troubleshooting information contains topics that can help you isolate and identify problems with DB2 diagnostic tools and utilities. There are also solutions to some of the most common problems and advice on how to solve problems you might encounter with your DB2 database products.

### IBM Support Portal

See the IBM Support Portal if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the IBM Support Portal at [http://www.ibm.com/support/entry/portal/Overview/Software/Information\\_Management/DB2\\_for\\_Linux\\_UNIX\\_and\\_Windows](http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux_UNIX_and_Windows).

---

## Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

**Personal use:** You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

**Commercial use:** You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved.



You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.



---

## Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
1623-14, Shimotsuruma, Yamato-shi  
Kanagawa 242-8502 Japan

**The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

## Notices

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited  
U59/3600  
3600 Steeles Avenue East  
Markham, Ontario L3R 9Z7  
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *\_enter the year or years\_*. All rights reserved.

## **Trademarks**

IBM, the IBM logo, and [ibm.com](http://ibm.com)<sup>®</sup> are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



---

# Index

## A

about this book  
    Call Level Interface Guide and Reference, Volume 1 ix  
AllowGetDataColumnReaccess CLI/ODBC configuration  
    keyword 325  
AllowInterleavedGetData CLI/ODBC configuration  
    keyword 326  
AltHostName CLI/ODBC keyword 326  
AltPort CLI/ODBC keyword 327  
AppendAPIName CLI/ODBC configuration keyword 327  
AppendForFetchOnly CLI/ODBC configuration keyword 328  
AppendRowColToErrorMessage CLI/ODBC configuration  
    keyword 328  
AppUsesLobLocator CLI/ODBC configuration keyword 327  
ArrayInputChain CLI/ODBC configuration keyword 329  
AsyncEnable CLI/ODBC configuration keyword 330  
Attach CLI/ODBC configuration keyword 330  
attributes  
    connection 427  
    environment 427  
    querying 427  
    setting 427  
    statement  
        CLI 427  
Authentication CLI/ODBC keyword 331  
AutoCommit CLI/ODBC configuration keyword 332

## B

BIDI CLI/ODBC keyword 333  
BIGINT data type  
    conversion to C 520  
    display size 536  
    length 535  
    precision 532  
    scale 534  
BINARY data type  
    conversion to C 520  
    display size 536  
    length 535  
    precision 532  
    scale 534  
bind packages CLI function 76  
binding  
    application variables 106  
    array of columns 106  
    column bindings 10  
    file references  
        LOB columns 16  
        LOB parameters 19  
    parameter markers  
        function 22  
BitData CLI/ODBC configuration keyword 333  
BLOB data type  
    conversion to C 520  
    display size 536  
    length 535  
    precision 532  
    scale 534  
BlockForNRows CLI/ODBC configuration keyword 333

BlockLobs CLI/ODBC configuration keyword 334  
books  
    ordering 542  
bulk operations CLI function 41

## C

C language  
    data types 512  
call level interface (CLI)  
    compound SQL (CLI) statements  
        return codes 315  
    configuration  
        keywords 317  
    diagnostics overview 313  
    functions  
        summary 1  
        supported 178  
        Unicode 5  
    handles  
        allocating 7  
    initializing 427  
    keywords 317  
    options 427  
    Unicode  
        functions 5  
        vendor escape clauses 229  
CHAR data type  
    conversion to C 520  
    display size 536  
    length 535  
    precision 532  
    scale 534  
CheckForFork CLI/ODBC configuration keyword 335  
CLI function details  
    SQLColumns 63  
    SQLProcedures 252  
CLI functions  
    return codeSQLSTATE 313  
CLI/ODBC configuration keywords  
    SchemaList 396  
CLI/ODBC keywords  
    AllowGetDataColumnReaccess 325  
    AllowInterleavedGetData 326  
    AltHostName 326  
    AltPort 327  
    AppendAPIName 327  
    AppendForFetchOnly 328  
    AppendRowColToErrorMessage 328  
    AppUsesLobLocator 327  
    ArrayInputChain 329  
    AsyncEnable 330  
    Attach 330  
    Authentication 331  
    AutoCommit 332  
    BIDI 333  
    BitData 333  
    BlockForNRows 333  
    BlockLobs 334  
    CheckForFork 335  
    ClientAcctStr 336

CLI/ODBC keywords (*continued*)

ClientAppName 336  
 ClientBuffersUnboundLOBS 337  
 ClientEncAlg 337  
 ClientUserID 338  
 ClientWrkStnName 339  
 CLIPkg 335  
 ConcurrentAccessResolution 340  
 ConnectNode 341  
 ConnectTimeout 342  
 ConnectType 343  
 CurrentFunctionPath 343  
 CurrentImplicitXMLParseOption 344  
 CurrentMaintainedTableTypesForOpt 344  
 CURRENTOPTIMIZATIONPROFILE 345  
 CurrentPackagePath 345  
 CurrentPackageSet 346  
 CurrentRefreshAge 346  
 CurrentSchema 347  
 CurrentSQLID 347  
 CursorHold 347  
 CursorTypes 348  
 Database 352  
 DateTimeStringFormat 352  
 DB2Degree 349  
 DB2Explain 349  
 DB2NETNamedParam 350  
 DB2Optimization 350  
 DBAlias 351  
 DBName 351  
 DecimalFloatRoundingMode 353  
 DeferredPrepare 354  
 DescribeCall 355  
 DescribeInputOnPrepare 355  
 DescribeOutputLevel 356  
 DescribeParam 357  
 DiagLevel 358  
 DiagPath 358  
 DisableKeysetCursor 358  
 DisableMultiThread 359  
 DisableUnicode 359  
 DSN 352  
 EnableNamedParameterSupport 359  
 FET\_BUF\_SIZE 360  
 FileDSN 360  
 FloatPrecRadix 361  
 GetDataLobNoTotal 361  
 GranteeList 362  
 GrantorList 362  
 Graphic 363  
 Hostname 363  
 IgnoreWarnings 364  
 IgnoreWarnList 364  
 initialization file 322  
 Instance 364  
 Interrupt 365  
 KeepDynamic 366  
 KRBPlugin 365  
 listing by category 317  
 LoadXAInterceptor 368  
 LOBCacheSize 366  
 LOBFileThreshold 367  
 LOBMaxColumnSize 367  
 LockTimeout 368  
 LongDataCompat 368  
 MapBigintCDefault 369  
 MapCharToWChar 369

CLI/ODBC keywords (*continued*)

MapDateCDefault 370  
 MapDateDescribe 370  
 MapDecimalFloatDescribe 371  
 MapGraphicDescribe 372  
 MapTimeCDefault 372  
 MapTimeDescribe 373  
 MapTimestampCDefault 374  
 MapTimestampDescribe 374  
 MapXMLCDefault 375  
 MapXMLDescribe 375  
 MaxLOBBlockSize CLI/ODBC keyword 376  
 Mode 376  
 NotifyLevel 377  
 OleDbReportIsLongForLongTypes 377  
 OleDbReturnCharAsWChar 377  
 OleDbSQLColumnsSortByOrdinal 378  
 OnlyUseBigPackages 379  
 OptimizeForNRows 379  
 Patch1 380  
 Patch2 383  
 Port 386  
 ProgramID 387  
 ProgramName 387  
 PromoteLONGVARtoLOB 388  
 Protocol 388  
 PWD 379  
 PWDPlugin 380  
 QueryTimeoutInterval 389  
 ReadCommonSectionOnNullConnect 390  
 ReceiveTimeout 390  
 Reopt 390  
 ReportPublicPrivileges 391  
 ReportRetryErrorsAsWarnings 391  
 RetCatalogAsCurrServer 392  
 RetOleDbConnStr 392  
 RetryOnError 393  
 ReturnAliases 394  
 ReturnSynonymSchema 394  
 SaveFile 396  
 security 397  
 ServerMsgMask 398  
 ServerMsgTextSP 424  
 ServiceName 398  
 SkipTrace 399  
 SQLCODEMAP 399  
 SQLOverrideFileName 395  
 SSLClientKeystash 400  
 SSLClientKeystoredb 400  
 SSLClientKeystoreDBPassword 401  
 SSLClientLabel 399  
 StaticCapFile 401  
 StaticLogFile 402  
 StaticMode 402  
 StaticPackage 402  
 StmtConcentrator 403  
 StreamGetData 403  
 StreamPutData 404  
 SysSchema 404  
 TableType 405  
 TargetPrincipal 406  
 TempDir 406  
 TimestampTruncErrToWarning 407  
 Trace 407  
 TraceAPIList 408  
 TraceAPIList! 410  
 TraceComm 412



- CLI/ODBC keywords (*continued*)
  - TraceErrImmediate 413
  - TraceFileName 413
  - TraceFlush 414
  - TraceFlushOnError 415
  - TraceLocks 415
  - TracePathName 417
  - TracePIDList 416
  - TracePIDTID 417
  - TraceRefreshInterval 418
  - TraceStmtOnly 418
  - TraceTime 419
  - TraceTimestamp 419
  - Trusted\_Connection 420
  - TxnIsolation 421
  - UID 422
  - Underscore 422
  - UseOldStpCall 423
  - UseServerMsgSP 423
  - WarningList 424
  - XMLDeclaration 425
- ClientAcctStr CLI/ODBC configuration keyword 336
- ClientAppName CLI/ODBC configuration keyword 336
- ClientBuffersUnboundLOBS CLI/ODBC configuration keyword 337
- ClientEncAlgr CLI/ODBC configuration keyword 337
- ClientUserID CLI/ODBC configuration keyword 338
- ClientWrkStnName CLI/ODBC configuration keyword 339
- CLIPkg CLI/ODBC configuration keyword 335
- CLOB data type
  - conversion to C 520
  - display size 536
  - length 535
  - precision 532
  - scale 534
- closing cursor CLI function 48
- columns
  - attributes 50
  - CLI column attribute function 50
  - data retrieval 156
  - obtaining list 63
  - obtaining list and privileges 59
- ColumnwiseMRI CLI/ODBC configuration keyword 339
- ColumnwiseMRI keywords
  - PWD 339
- CommitOnEOF CLI/ODBC configuration keyword 340
- CommitOnEOF keywords
  - PWD 340
- compound SQL (CLI) statement
  - return codes 315
- ConcurrentAccessResolution CLI/ODBC configuration keyword 340
- connection handles
  - allocating 7
  - freeing 146
- connections
  - attributes
    - changing 427
    - determining 36
    - getting 151
    - list 436
    - setting 264
  - connection strings 427
  - SQLConnect function 69
  - SQLDriverConnect function 88
  - switching in mixed applications 267
- ConnectNode CLI/ODBC configuration keyword 341
- ConnectTimeout CLI/ODBC configuration keyword details 342
- ConnectType CLI/ODBC configuration keyword 343
  - conversion
    - CLI applications
      - C to SQL data types 527
      - display sizes of SQL data types 536
      - lengths of SQL data types 535
      - precisions of SQL data types 532
      - scales of SQL data types 534
      - SQL to C data types 520
      - summary 517
      - summary of SQL data types 511
    - data types in CLI 517
  - copying descriptors CLI function 71
- CurrentFunctionPath CLI/ODBC configuration keyword 343
- CurrentImplicitXMLParseOption CLI/ODBC configuration keyword 344
- CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword 344
- CURRENTOPTIMIZATIONPROFILE CLI/ODBC configuration parameter 345
- CurrentPackagePath CLI/ODBC configuration keyword 345
- CurrentPackageSet CLI/ODBC configuration keyword 346
- CurrentRefreshAge CLI/ODBC configuration keyword 346
- CurrentSchema CLI/ODBC configuration keyword 347
- CurrentSQLID CLI/ODBC configuration keyword 347
- CursorHold CLI/ODBC configuration keyword 347
- cursors
  - call level interface (CLI)
    - closing 48
  - names
    - getting 154
    - setting 269
  - positioning
    - rules for SQLFetchScroll 138
- CursorTypes CLI/ODBC configuration keyword 348

## D

- data sources
  - connecting to
    - SQLBrowseConnect function 36
    - SQLConnect function 69
    - SQLDriverConnect function 88
  - disconnecting from using CLI function 86
- data types
  - C
    - CLI applications 511, 512
    - conversion
      - CLI 517
    - SQL
      - CLI applications 511
      - supported by database management systems 222
- Database CLI/ODBC configuration keyword 352
- database systems
  - retrieving information about 180
- databases
  - creating
    - SQLCreateDb function 74
  - retrieving list 77
- DATE data type
  - SQL
    - conversion to C 520
    - display size 536
    - length 535
    - precision 532

- DATE data type *(continued)*
  - SQL *(continued)*
    - scale 534
- DateTimeStringFormat CLI/ODBC configuration
  - keyword 352
- DB2 Information Center
  - languages 544
  - updating 545, 546
  - versions 543
- db2cli.ini file
  - attributes 427
  - details 322
- DB2Degree CLI/ODBC configuration keyword 349
- DB2Explain CLI/ODBC configuration keyword 349
- DB2NETNamedParam CLI/ODBC configuration
  - keyword 350
- DB2NODE environment variable
  - ConnectNode CLI/ODBC configuration keyword
    - impact 341
- DB2Optimization CLI/ODBC configuration keyword 350
- DBAlias CLI/ODBC configuration keyword 351
- DBCLOB data type
  - conversion to C 520
  - display size 536
  - length 535
  - precision 532
  - scale 534
- DBName CLI/ODBC configuration keyword 351
- DECIMAL data type
  - conversion
    - C/C++ 520
  - display size 536
  - length 535
  - precision 532
  - scale 534
- DecimalFloatRoundingMode CLI/ODBC configuration
  - keyword 353
- DeferredPrepare CLI/ODBC configuration keyword 354
- deprecated functionality
  - CLI functions
    - SQLAllocConnect 6
    - SQLAllocEnv 7
    - SQLAllocStmt 10
    - SQLColAttributes 58
    - SQLError 98
    - SQLExtendedFetch 109
    - SQLFreeConnect 145
    - SQLFreeEnv 146
    - SQLGetConnectOption 154
    - SQLGetSQLCA 216
    - SQLGetStmtOption 219
    - SQLParamOptions 238
    - SQLSetColAttributes 264
    - SQLSetConnectOption 269
    - SQLSetParam 281
    - SQLSetStmtOption 293
    - SQLTransact 312
- DescribeCall CLI/ODBC configuration keyword 355
- DescribeInputOnPrepare CLI/ODBC configuration
  - keyword 355
- DescribeOutputLevel CLI/ODBC configuration keyword 356
- DescribeParam CLI/ODBC configuration keyword 357
- descriptor handles
  - allocating 7
  - freeing 146
- descriptors
  - copying
    - SQLCopyDesc function 71
  - FieldIdentifier argument values 489
  - values
    - getting from multiple fields 166
    - getting from single field 162
    - header fields 489, 500
    - record fields 489, 500
    - setting for multiple fields 276
    - setting for single field 272
- details
  - SQLReloadConfig function 259
- DiagIdentifier argument 505
- DiagLevel CLI/ODBC keyword 358
- diagnostic information
  - CLI applications 313
  - diagnostic data structures
    - getting value from single field 170
    - getting values from multiple fields 174
- DiagPath CLI/ODBC keyword 358
- DisableKeysetCursor CLI/ODBC configuration keyword 358
- DisableMultiThread CLI/ODBC configuration keyword 359
- DisableUnicode CLI/ODBC configuration keyword 359
- disconnect from a data source CLI function 86
- documentation
  - overview 539
  - PDF files 539
  - printed 539
  - terms and conditions of use 548
- DOUBLE data type
  - conversion to C 520
  - display size 536
  - length 535
  - precision 532
  - scale 534
- drop a database CLI function 93
- DSN CLI/ODBC keyword 352

## E

- EnableNamedParameterSupport CLI/ODBC configuration
  - keyword 359
- environment attributes
  - changing 427
  - obtaining current 177
  - setting 280
- environment handles
  - allocating 7
  - freeing 146

## F

- FET\_BUF\_SIZE CLI/ODBC configuration keyword 360
- fetching
  - next row CLI function 125
  - rowset CLI function 132
- File DSN
  - database to connect 352
  - host name 363
  - IP address 363
  - protocol used 388
  - service name 398
- FileDSN CLI/ODBC keyword 360
- FLOAT data type
  - conversion to C 520

FLOAT data type (*continued*)  
  display size 536  
  length 535  
  precision 532  
  scale 534  
FloatPrecRadix CLI/ODBC configuration keyword 361  
foreign keys  
  getting list of columns 141  
freeing CLI handles  
  SQLFreeHandle function 146  
  SQLFreeStmt function 149  
functions  
  querying whether supported 178

## G

GetDataLobNoTotal CLI/ODBC configuration keyword 361  
GranteeList CLI/ODBC configuration keyword 362  
GrantorList CLI/ODBC configuration keyword 362  
Graphic CLI/ODBC configuration keyword 363  
GRAPHIC data type  
  conversion to C 520  
  display size 536  
  length 535  
  precision 532  
  scale 534

## H

handles  
  freeing  
    SQLFreeHandle function 146  
help  
  configuring language 544  
  SQL statements 543  
Hostname CLI/ODBC configuration keyword 363

## I

IgnoreWarnings CLI/ODBC configuration keyword 364  
IgnoreWarnList CLI/ODBC configuration keyword 364  
IN DATABASE statement 351  
indexes  
  statistics  
    obtaining 299  
INI file 322  
Instance CLI/ODBC keyword 364  
INTEGER data type  
  conversion to C 520  
  display size 536  
  length 535  
  precision 532  
  scale 534  
Interrupt CLI/ODBC keyword 365  
INVALID\_HANDLE return code 313

## K

KeepDynamic CLI/ODBC configuration keyword 366  
KRBPlugin CLI/ODBC keyword 365

## L

large objects (LOBs)  
  length 210

large objects (LOBs) (*continued*)  
  obtaining portion of value 219  
LoadXAInterceptor CLI/ODBC configuration keyword 368  
LOBCacheSize CLI/ODBC configuration keyword 366  
LOBFileThreshold CLI/ODBC configuration keyword 367  
LOBMaxColumnSize CLI/ODBC configuration keyword 367  
LockTimeout CLI/ODBC configuration keyword 368  
LongDataCompat CLI/ODBC configuration keyword  
  details 368  
LONGVARBINARY data type  
  conversion to C 520  
  display size 536  
  length 535  
  precision 532  
  scale 534  
LONGVARCHAR data type  
  conversion to C 520  
  display size 536  
  length 535  
  precision 532  
  scale 534  
LONGVARGRAPHIC data type  
  conversion to C 520  
  display size 536  
  length 535  
  precision 532  
  scale 534

## M

MapBigintCDefault CLI/ODBC configuration keyword 369  
MapCharToWChar CLI/ODBC configuration keyword 369  
MapDateCDefault CLI/ODBC configuration keyword 370  
MapDateDescribe CLI/ODBC configuration keyword 370  
MapDecimalFloatDescribe CLI/ODBC configuration  
  keyword 371  
MapGraphicDescribe CLI/ODBC configuration keyword 372  
MapTimeCDefault CLI/ODBC configuration keyword 372  
MapTimeDescribe CLI/ODBC configuration keyword 373  
MapTimestampCDefault CLI/ODBC configuration  
  keyword 374  
MapTimestampDescribe CLI/ODBC configuration  
  keyword 374  
MapXMLCDefault CLI/ODBC configuration keyword 375  
MapXMLDescribe CLI/ODBC configuration keyword 375  
MaxLOBBlockSize CLI/ODBC configuration keyword 376  
Mode CLI/ODBC configuration keyword 376  
more result sets CLI function 227

## N

native error codes 314  
native SQL text CLI function 229  
notices 551  
NotifyLevel CLI/ODBC keyword 377  
NUMERIC data type  
  conversion to C 520  
  display size 536  
  length 535  
  precision 532  
  scale 534

## O

OleDbReportIsLongForLongTypes CLI/ODBC configuration  
  keyword 377

OleDbReturnCharAsWChar CLI/ODBC configuration keyword 377  
OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword 378  
OnlyUseBigPackages CLI/ODBC configuration keyword 379  
OptimizeForNRRows CLI/ODBC configuration keyword 379  
ordering DB2 books 542

## P

packages  
  binding  
    SQLCreatePkg function 76  
parallelism  
  degree 349  
parameter markers  
  obtaining description 84  
  obtaining number 230  
parameters  
  data value passing 256  
  getting next 236  
  input/output  
    obtaining information 119, 246  
Patch1 CLI/ODBC configuration keyword 380  
Patch2 CLI/ODBC configuration keyword 383  
port CLI/ODBC configuration keyword 386  
precision  
  SQL data types 532  
prepared SQL statements  
  CLI applications  
    extended 110  
    syntax 239  
primary keys  
  columns  
    obtaining using CLI function 243  
problem determination  
  information available 548  
  tutorials 548  
procedures  
  names  
    obtaining list 115, 252  
ProgramID CLI/ODBC configuration keyword 387  
ProgramName CLI/ODBC configuration keyword 387  
PromoteLONGVARtoLOB CLI/ODBC configuration keyword 388  
Protocol CLI/ODBC configuration keyword 388  
PWD CLI/ODBC configuration keyword 379  
PWDPlugin CLI/ODBC keyword 380

## Q

QueryTimeoutInterval CLI/ODBC configuration keyword 389

## R

ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword 390  
REAL SQL data type  
  conversion  
    to C data type 520  
  display size 536  
  length 535  
  precision 532  
  scale 534  
ReceiveTimeout CLI/ODBC configuration keyword 390

Reopt CLI/ODBC configuration keyword 390  
ReportPublicPrivileges CLI/ODBC configuration keyword 391  
ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword 391  
result columns  
  getting number 234  
result sets  
  associating with handle 232  
  CLI  
    SQLMoreResults function 227  
RetCatalogAsCurrServer CLI/ODBC configuration keyword 392  
RetOleDbConnStr CLI/ODBC configuration keyword 392  
RetryOnError CLI/ODBC configuration keyword 393  
return codes  
  CLI  
    compound SQL 315  
    functions 313  
ReturnAliases CLI/ODBC configuration keyword 394  
ReturnSynonymSchema CLI/ODBC configuration keyword 394  
row identifiers  
  getting information by using CLI function 294  
row sets  
  CLI functions  
    fetching 132  
    setting cursor position 282  
rows  
  count retrieval  
    SQLRowCount function 262

## S

SaveFile CLI/ODBC keyword 396  
scale  
  SQL data types 534  
SchemaList CLI/ODBC configuration keyword 396  
security configuration parameter for CLI/ODBC applications 397  
ServerMsgMask CLI/ODBC configuration keyword 398  
ServerMsgTextSP CLI/ODBC configuration keyword 424  
ServiceName CLI/ODBC configuration keyword 398  
SET CURRENT SCHEMA statement 347  
SkipTrace CLI/ODBC configuration keyword 399  
SMALLINT data type  
  conversion to C/C++ 520  
  display size 536  
  length 535  
  precision 532  
  scale 534  
SQL data types  
  display size 536  
  lengths 535  
  precision 532  
  scale 534  
SQL statements  
  help  
    displaying 543  
SQL\_  
  ROWSET\_SIZE statement attribute 466  
SQL\_ATTR\_  
  ACCESS\_MODE connection attribute 436  
  ALLOW\_INTERLEAVED\_GETDATA  
    AllowInterleavedGetData CLI/ODBC configuration keyword 326  
    connection attribute 436

## SQL\_ATTR\_ (continued)

ALLOW\_INTERLEAVED\_GETDATA (continued)  
 statement attribute 466

ANSI\_APP connection attribute 436

APP\_PARAM\_DESC statement attribute 466

APP\_ROW\_DESC statement attribute 466

APP\_USES\_LOB\_LOCATOR  
 AppUsesLOBLocator CLI/ODBC configuration  
 keyword 327  
 connection attribute 436  
 statement attribute 466

APPEND\_FOR\_FETCH\_ONLY  
 AppendForFetchOnly CLI/ODBC configuration  
 keyword 328  
 connection attribute 436

ASYNC\_ENABLE  
 AsyncEnable CLI/ODBC configuration keyword 330  
 connection attribute 436  
 statement attribute 466

AUTO\_IPD  
 connection attribute 436

AUTOCOMMIT  
 AutoCommit CLI/ODBC configuration keyword 332  
 connection attribute 436

BLOCK\_FOR\_NROWS statement attribute 466

BLOCK\_LOBS  
 BlockLobs CLI/ODBC configuration keyword 334  
 statement attribute 466

CALL\_RETURN statement attribute 466

CHAINING\_BEGIN statement attribute 466

CHAINING\_END statement attribute 466

CLIENT\_ENCALG 337

CLIENT\_LOB\_BUFFERING  
 connection attribute 436  
 statement attribute 466

CLOSE\_BEHAVIOR statement attribute 466

CLOSEOPEN statement attribute 466

COLUMNWISE\_MRI  
 connection attribute 436  
 statement attribute 466

COMMITONEOF  
 connection attribute 436

CONCURRENCY statement attribute 466

CONCURRENT\_ACCESS\_RESOLUTION  
 ConcurrentAccessResolution CLI/ODBC configuration  
 keyword 340  
 connection attribute 436

CONN\_CONTEXT connection attribute 436

CONNECT\_NODE  
 connection attribute 436  
 ConnectNode CLI/ODBC configuration keyword 341

CONNECTION\_DEAD connection attribute 436

CONNECTION\_POOLING environment attribute 429

CONNECTION\_TIMEOUT connection attribute 436

CONNECTTYPE  
 connection attribute 436  
 ConnectType CLI/ODBC configuration keyword 343  
 environment attribute 429

CP\_MATCH environment attribute 429

CURRENT\_CATALOG connection attribute 436

CURRENT\_IMPLICIT\_XMLPARSE\_OPTION connection  
 attribute 436

CURRENT\_PACKAGE\_PATH  
 connection attribute 436  
 CurrentPackagePath CLI/ODBC configuration  
 keyword 345

## SQL\_ATTR\_ (continued)

CURRENT\_PACKAGE\_SET  
 connection attribute 436  
 CurrentPackageSet CLI/ODBC configuration  
 keyword 346

CURRENT\_SCHEMA connection attribute 436

CURSOR\_HOLD  
 CursorHold CLI/ODBC configuration keyword 347  
 statement attribute 466

CURSOR\_SCROLLABLE statement attribute 466

CURSOR\_SENSITIVITY statement attribute 466

CURSOR\_TYPE statement attribute 466

DB2\_APPLICATION\_HANDLE connection attribute 436

DB2\_APPLICATION\_ID connection attribute 436

DB2\_NOBINDOUT statement attribute 466

DB2\_SQLERRP connection attribute 436

DB2ESTIMATE connection attribute 436

DB2EXPLAIN  
 connection attribute 436  
 DB2Explain CLI/ODBC configuration keyword 349

DECFLOAT\_ROUNDING\_MODE  
 connection attribute 436  
 DecimalFloatRoundingMode CLI/ODBC configuration  
 keyword 353

DEFERRED\_PREPARE  
 DeferredPrepare CLI/ODBC configuration  
 keyword 354  
 statement attribute 466

DESCRIBE\_CALL  
 connection attribute 436  
 DescribeCall CLI/ODBC configuration keyword 355

DESCRIBE\_OUTPUT\_LEVEL 356  
 connection attribute 436

DIAGLEVEL environment attribute 429

DIAGPATH environment attribute 429

EARLYCLOSE  
 connection attribute 436  
 EARLYCLOSE statement attribute 466

ENABLE\_AUTO\_IPD statement attribute 466

ENLIST\_IN\_DTC connection attribute 436

FET\_BUF\_SIZE  
 FET\_BUF\_SIZE CLI/ODBC configuration keyword 360

FETCH\_BOOKMARK\_PTR statement attribute 466

FREE\_LOCATORS\_ON\_FETCH connection attribute 436

GET\_LATEST\_MEMBER  
 connection attribute 436

IMP\_PARAM\_DESC statement attribute 466

IMP\_ROW\_DESC statement attribute 466

INFO\_ACCTSTR  
 ClientAcctStr CLI/ODBC configuration keyword 336  
 connection attribute 436  
 environment attribute 429

INFO\_APPLNAME  
 ClientAppName CLI/ODBC configuration  
 keyword 336  
 connection attribute 436  
 environment attribute 429

INFO\_PROGRAMID  
 ProgramID CLI/ODBC configuration keyword 387

INFO\_PROGRAMID connection attribute 436

INFO\_PROGRAMNAME  
 connection attribute 436  
 ProgramName CLI/ODBC configuration keyword 387

INFO\_USERID  
 ClientUserID CLI/ODBC configuration keyword 338  
 connection attribute 436  
 environment attribute 429

SQL\_ATTR\_ (continued)

- INFO\_WRKSTNNAME
  - ClientWrkStnName CLI/ODBC configuration keyword 339
  - connection attribute 436
  - environment attribute 429
- INSERT\_BUFFERING statement attribute 466
- KEEP\_DYNAMIC
  - connection attribute 436
  - KeepDynamic CLI/ODBC configuration keyword 366
- KEYSET\_SIZE statement attribute 466
- LOAD\_INFO statement attribute 466
- LOAD\_ROWS\_COMMITTED\_PTR statement attribute 466
- LOAD\_ROWS\_DELETED\_PTR statement attribute 466
- LOAD\_ROWS\_LOADED\_PTR statement attribute 466
- LOAD\_ROWS\_READ\_PTR statement attribute 466
- LOAD\_ROWS\_REJECTED\_PTR statement attribute 466
- LOAD\_ROWS\_SKIPPED\_PTR statement attribute 466
- LOB\_CACHE\_SIZE
  - connection attribute 436
  - LOBCacheSize CLI/ODBC configuration keyword 366
  - statement attribute 466
- LOGIN\_TIMEOUT
  - connection attribute 436
  - ConnectTimeout CLI/ODBC configuration keyword 342
- LONGDATA\_COMPAT
  - connection attribute 436
  - LongDataCompat CLI/ODBC configuration keyword 368
- MAPCHAR
  - connection attribute 436
  - MapCharToWChar CLI/ODBC configuration keyword 369
- MAX\_LENGTH
  - statement attribute 466
- MAX\_LOB\_BLOCK\_SIZE
  - connection attribute 436
  - MaxLOBBlockSize CLI/ODBC configuration keyword 376
  - statement attribute 466
- MAX\_ROWS statement attribute 466
- MAXCONN
  - connection attribute 436
  - environment attribute 429
- METADATA\_ID
  - connection attribute 436
  - statement attribute 466
- MODIFIED\_BY statement attribute 466
- NOSCAN statement attribute 466
- NOTIFYLEVEL environment attribute 429
- ODBC\_CURSORS connection attribute 436
- ODBC\_VERSION environment attribute 429
- OPTIMIZE\_FOR\_NROWS
  - OptimizeForNRows CLI/ODBC configuration keyword 379
  - statement attribute 466
- OPTIMIZE\_SQLCOLUMNS statement attribute 466
- OUTPUT\_NTS 429
- OVERRIDE\_CODEPAGE
  - connection attribute 436
- PACKET\_SIZE connection attribute 436
- PARAM\_BIND\_OFFSET\_PTR statement attribute 466
- PARAM\_BIND\_TYPE statement attribute 466
- PARAM\_OPERATION\_PTR statement attribute 466
- PARAM\_STATUS\_PTR statement attribute 466
- PARAMOPT\_ATOMIC statement attribute 466

SQL\_ATTR\_ (continued)

- PARAMS\_PROCESSED\_PTR statement attribute 466
- PARAMSET\_SIZE statement attribute 466
- PARC\_BATCH
  - connection attribute 436
- PING\_DB connection attribute 436
- PING\_NTICES connection attribute 436
- PING\_REQUEST\_PACKET\_SIZE connection attribute 436
- PREFETCH statement attribute 466
- PROCESSCTRL
  - CheckForFork CLI/ODBC configuration keyword 335
  - environment attribute 429
- QUERY\_OPTIMIZATION\_LEVEL statement attribute 466
- QUERY\_TIMEOUT
  - QueryTimeoutInterval CLI/ODBC configuration keyword 389
  - statement attribute 466
- QUIET\_MODE connection attribute 436
- RECEIVE\_TIMEOUT
  - connection attribute 436
  - ReceiveTimeout CLI/ODBC configuration keyword 390
- REOPT
  - connection attribute 436
  - Reopt CLI/ODBC configuration keyword 390
  - statement attribute 466
- REPORT\_ISLONG\_FOR\_LONGTYPES\_OLEDB
  - connection attribute 436
  - OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword 377
- REPORT\_SEAMLESSFAILOVER\_WARNING
  - connection attribute 436
- REPORT\_TIMESTAMP\_TRUNC\_AS\_WARN connection attribute 436
- RESET\_CONNECTION
  - environment attribute 429
- RETRIEVE\_DATA statement attribute 466
- RETURN\_USER\_DEFINED\_TYPES
  - statement attribute 466
- ROW\_ARRAY\_SIZE statement attribute 466
- ROW\_BIND\_OFFSET\_PTR statement attribute 466
- ROW\_BIND\_TYPE statement attribute 466
- ROW\_NUMBER statement attribute 466
- ROW\_OPERATION\_PTR statement attribute 466
- ROW\_STATUS\_PTR statement attribute 466
- ROWCOUNT\_PREFETCH
  - statement attribute 466
- ROWS\_FETCHED\_PTR statement attribute 466
- SERVER\_MSGTXT\_MASK
  - connection attribute 436
  - ServerMsgMask CLI/ODBC configuration keyword 398
- SERVER\_MSGTXT\_SP
  - connection attribute 436
  - ServerMsgTextSP CLI/ODBC configuration keyword 424
  - UseServerMsgSP CLI/ODBC configuration keyword 423
- SESSION\_TIME\_ZONE
  - connection attribute 436
- SIMULATE\_CURSOR statement attribute 466
- SQLCOLUMNS\_SORT\_BY\_ORDINAL\_OLEDB
  - connection attribute 436
  - OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword 378
- STMT\_CONCENTRATOR
  - connection attribute 436

## SQL\_ATTR\_ (continued)

STMT\_CONCENTRATOR (continued)  
     statement attribute 466  
     StmtConcentrator CLI/ODBC configuration  
         keyword 403  
 STMTXN\_ISOLATION statement attribute 466  
 STREAM\_GETDATA  
     connection attribute 436  
     statement attribute 466  
     StreamGetData CLI/ODBC configuration keyword 403  
 SYNC\_POINT  
     connection attribute 436  
     environment attribute 429  
 TRACE  
     connection attribute 436  
     environment attribute 429  
     Trace CLI/ODBC configuration keyword 407  
 TRACEFILE connection attribute 436  
 TRACENOHEADER environment attribute 429  
 TRANSLATE\_LIB connection attribute 436  
 TRANSLATE\_OPTION connection attribute 436  
 TRUSTED\_CONTEXT\_PASSWORD  
     connection attribute 436  
 TRUSTED\_CONTEXT\_USERID  
     connection attribute 436  
 TXN\_ISOLATION  
     connection attribute 436  
     statement attribute 466  
     TxnIsolation CLI/ODBC configuration keyword 421  
 USE\_2BYTES\_OCTET\_LENGTH environment  
     attribute 429  
 USE\_BOOKMARKS statement attribute 466  
 USE\_LIGHT\_INPUT\_SQLDA environment attribute 429  
 USE\_LIGHT\_OUTPUT\_SQLDA environment attribute 429  
 USE\_LOAD\_API statement attribute 466  
 USE\_TRUSTED\_CONTEXT  
     connection attribute 436  
 USER\_REGISTRY\_NAME  
     connection attribute 436  
     environment attribute 429  
 WCHARTYPE connection attribute 436  
 XML\_DECLARATION  
     connection attribute 436  
     statement attribute 466  
 XQUERY\_STATEMENT statement attribute 466  
 SQL\_C\_BINARY data type 527  
 SQL\_C\_BIT data type 527  
 SQL\_C\_CHAR 527  
 SQL\_C\_DATE data type 527  
 SQL\_C\_DBCHAR data type 527  
 SQL\_C\_DOUBLE data type 527  
 SQL\_C\_FLOAT data type 527  
 SQL\_C\_LONG data type 527  
 SQL\_C\_SHORT data type 527  
 SQL\_C\_TIME data type 527  
 SQL\_C\_TIMESTAMP data type 527  
 SQL\_C\_TIMESTAMP\_EXT data type 527  
 SQL\_C\_TINYINT data type 527  
 SQL\_DESC\_  
     ALLOC\_TYPE  
         details 489  
         initialization value 500  
     ARRAY\_SIZE  
         details 489  
         initialization value 500  
     ARRAY\_STATUS\_PTR  
         details 489

## SQL\_DESC\_ (continued)

ARRAY\_STATUS\_PTR (continued)  
     initialization value 500  
 AUTO\_UNIQUE\_VALUE  
     details 50, 489  
     initialization value 500  
 BASE\_COLUMN\_NAME 489  
     details 50  
     initialization value 500  
 BASE\_TABLE\_NAME  
     details 50, 489  
     initialization value 500  
 BIND\_OFFSET\_PTR  
     details 489  
     initialization value 500  
 BIND\_TYPE  
     details 489  
     initialization value 500  
 CASE\_SENSITIVE  
     details 50, 489  
     initialization value 500  
 CATALOG\_NAME  
     details 50, 489  
     initialization value 500  
 CONCISE\_TYPE  
     details 50, 489  
     initialization value 500  
 COUNT 50  
 COUNT\_ALL 489  
 DATA\_PTR  
     details 489  
     initialization value 500  
 DATETIME\_INTERVAL\_CODE  
     details 489  
     initialization value 500  
 DATETIME\_INTERVAL\_PRECISION  
     details 489  
     initialization value 500  
 DISPLAY\_SIZE  
     details 50, 489  
     initialization value 500  
 DISTINCT\_TYPE 50  
 FIXED\_PREC\_SCALE  
     details 50, 489  
     initialization value 500  
 INDICATOR\_PTR  
     details 489  
     initialization value 500  
 LABEL  
     details 50, 489  
 LENGTH  
     details 50, 489  
     initialization value 500  
 LITERAL\_PREFIX  
     details 50, 489  
     initialization value 500  
 LITERAL\_SUFFIX  
     details 50, 489  
     initialization value 500  
 LOCAL\_TYPE\_NAME  
     details 50, 489  
     initialization value 500  
 NAME  
     details 50, 489  
     initialization value 500  
 NULLABLE  
     details 50, 489

SQL\_DESC\_ (continued)  
   NULLABLE (continued)  
     initialization value 500  
   NUM\_PREC\_RADIX  
     details 489  
     initialization value 500  
   NUM\_PREX\_RADIX  
     details 50  
   OCTET\_LENGTH  
     details 50, 489  
     initialization value 500  
   OCTET\_LENGTH\_PTR  
     details 489  
     initialization value 500  
   PARAMETER\_TYPE  
     details 489  
     initialization value 500  
   PRECISION  
     details 50, 489  
     initialization value 500  
   ROWS\_PROCESSED\_PTR  
     details 489  
     initialization value 500  
   SCALE  
     details 50, 489  
     initialization value 500  
   SCHEMA\_NAME  
     details 50, 489  
     initialization value 500  
   SEARCHABLE  
     details 50, 489  
     initialization value 500  
   TABLE\_NAME  
     details 50, 489  
     initialization value 500  
   TYPE  
     details 50, 489  
     initialization value 500  
   TYPE\_NAME  
     details 50, 489  
     initialization value 500  
   UNNAMED  
     details 50, 489  
     initialization value 500  
   UNSIGNED  
     details 50, 489  
     initialization value 500  
   UPDATABLE  
     details 50, 489  
     initialization value 500  
 SQL\_DIAG\_  
   header fields 505  
   record fields 505  
 SQL\_ERROR return code 313  
 SQL\_NEED\_DATA return code 313  
 SQL\_NO\_DATA\_FOUND return code 313  
 SQL\_STILL\_EXECUTING return code 313  
 SQL\_SUCCESS return code 313  
 SQL\_SUCCESS\_WITH\_INFO return code 313  
 SQLAllocConnect deprecated CLI function 6  
 SQLAllocEnv deprecated CLI function 7  
 SQLAllocHandle CLI function 7  
 SQLAllocStmt deprecated CLI function 10  
 SQLBindCol CLI function  
   details 10  
 SQLBindFileToCol CLI function 16  
 SQLBindFileToParam CLI function 19  
 SQLBindParameter CLI function  
   details 22  
 SQLBrowseConnect CLI function  
   details 36  
   Unicode version 5  
 SQLBrowseConnectW CLI function 5  
 SQLBulkOperations CLI function  
   details 41  
 SQLCancel CLI function 46  
 SQLCloseCursor CLI function 48  
 SQLCODEMAP configuration parameter  
   details 399  
 SQLColAttribute CLI function  
   details 50  
   Unicode version 5  
 SQLColAttributes CLI function  
   deprecated 58  
   Unicode version 5  
 SQLColAttributesW CLI function 5  
 SQLColAttributeW CLI function 5  
 SQLColumnPrivileges CLI function  
   details 59  
   Unicode version 5  
 SQLColumnPrivilegesW CLI function 5  
 SQLColumns CLI function  
   Unicode version 5  
 SQLColumnsW CLI function 5  
 SQLConnect CLI function  
   details 69  
   Unicode version 5  
 SQLConnectW CLI function 5  
 SQLCopyDesc CLI function 71  
 SQLCreateDb CLI function 74  
   Unicode version 5  
 SQLCreateDbW CLI function 5  
 SQLCreatePkg CLI function 76  
 SQLDataSources CLI function  
   details 77  
   Unicode version 5  
 SQLDataSourcesW CLI function 5  
 SQLDescribeCol CLI function  
   details 80  
   Unicode version 5  
 SQLDescribeColW CLI function 5  
 SQLDescribeParam CLI function 84  
 SQLDisconnect CLI function 86  
 SQLDriverConnect CLI function  
   default values 427  
   details 88  
   Trusted\_connection CLI/ODBC configuration  
     keyword 420  
   Unicode version 5  
 SQLDriverConnectW CLI function 5  
 SQLDropDb CLI function 93  
   Unicode version 5  
 SQLDropDbW CLI function 5  
 SQLEndTran CLI function  
   details 95  
 SQLError deprecated CLI function  
   details 98  
   Unicode version 5  
 SQLErrorW CLI function 5  
 SQLExecDirect CLI function  
   details 98  
   Unicode version 5  
 SQLExecDirectW CLI function 5



SQLExecute CLI function  
     details 103  
 SQLExtendedBind CLI function 106  
 SQLExtendedFetch deprecated CLI function 109  
 SQLExtendedPrepare CLI function  
     details 110  
     Unicode version 5  
 SQLExtendedPrepareW CLI function 5  
 SQLExtendedProcedureColumns  
     Unicode version 5  
 SQLExtendedProcedureColumns CLI function  
     details 119  
 SQLExtendedProcedureColumnsW CLI function 5  
 SQLExtendedProcedures  
     Unicode version 5  
 SQLExtendedProcedures CLI function  
     details 115  
 SQLExtendedProceduresW CLI function 5  
 SQLFetch CLI function  
     details 125  
 SQLFetchScroll CLI function  
     cursor positioning rules 138  
     details 132  
 SQLForeignKeys CLI function  
     details 141  
     Unicode version 5  
 SQLForeignKeysW CLI function 5  
 SQLFreeConnect deprecated CLI function 145  
 SQLFreeEnv deprecated CLI function 146  
 SQLFreeHandle CLI function 146  
 SQLFreeStmt CLI function 149  
 SQLGetConnectAttr CLI function  
     details 151  
     Unicode version 5  
 SQLGetConnectAttrW CLI function 5  
 SQLGetConnectOption deprecated CLI function  
     details 154  
     Unicode version 5  
 SQLGetConnectOptionW CLI function 5  
 SQLGetCursorName CLI function  
     details 154  
     Unicode version 5  
 SQLGetCursorNameW CLI function 5  
 SQLGetData CLI function  
     details 156  
 SQLGetDescField CLI function  
     details 162  
     Unicode version 5  
 SQLGetDescFieldW CLI function 5  
 SQLGetDescRec CLI function  
     details 166  
     Unicode version 5  
 SQLGetDescRecW CLI function 5  
 SQLGetDiagField CLI function  
     details 170  
     Unicode version 5  
 SQLGetDiagFieldW CLI function 5  
 SQLGetDiagRec CLI function  
     details 174  
     Unicode version 5  
 SQLGetDiagRecW CLI function 5  
 SQLGetEnvAttr CLI function 177  
 SQLGetFunctions CLI function 178  
 SQLGetInfo CLI function  
     details 180  
     Unicode version 5  
 SQLGetInfoW CLI function 5  
 SQLGetLength CLI function 210  
 SQLGetPosition CLI function 212  
     Unicode version 5  
 SQLGetPositionW CLI function 5  
 SQLGetSQLCA deprecated CLI function 216  
 SQLGetStmtAttr CLI function  
     details 216  
     Unicode version 5  
 SQLGetStmtAttrW CLI function 5  
 SQLGetStmtOption deprecated CLI function 219  
 SQLGetSubString CLI function 219  
 SQLGetTypeInfo CLI function 222  
 SQLMoreResults CLI function 227  
 SQLNativeSql CLI function  
     details 229  
     Unicode version 5  
 SQLNativeSqlW CLI function 5  
 SQLNextResult CLI function 232  
 SQLNumParams CLI function 230  
 SQLNumResultCols CLI function  
     details 234  
 SQLOverrideFileName CLI/ODBC configuration  
     keyword 395  
 SQLParamData CLI function 236  
 SQLParamOptions deprecated CLI function 238  
 SQLPrepare CLI function  
     details 239  
     Unicode version 5  
 SQLPrepareW CLI function 5  
 SQLPrimaryKeys CLI function  
     details 243  
     Unicode version 5  
 SQLPrimaryKeysW CLI function 5  
 SQLProcedureColumns CLI function  
     details 246  
     Unicode version 5  
 SQLProcedureColumnsW CLI function 5  
 SQLProcedures CLI function  
     Unicode version 5  
 SQLProceduresW CLI function 5  
 SQLPutData CLI function 256  
 SQLReloadConfig CLI function  
     Unicode version 5  
 SQLReloadConfigW CLI function 5  
 SQLRowCount CLI function  
     details 262  
 SQLSetColAttributes deprecated CLI function 264  
 SQLSetConnectAttr CLI function  
     details 264  
     Unicode version 5  
 SQLSetConnectAttrW CLI function 5  
 SQLSetConnection CLI function 267  
 SQLSetConnectOption deprecated CLI function  
     details 269  
     Unicode version 5  
 SQLSetConnectOptionW CLI function 5  
 SQLSetCursorName CLI function  
     details 269  
     Unicode version 5  
 SQLSetCursorNameW CLI function 5  
 SQLSetDescField CLI function  
     details 272  
     Unicode version 5  
 SQLSetDescFieldW CLI function 5  
 SQLSetDescRec CLI function 276  
 SQLSetEnvAttr CLI function 280  
 SQLSetParam deprecated CLI function 281

- SQLSetPos CLI function 282
- SQLSetStmtAttr CLI function
  - details 289
  - Unicode version 5
- SQLSetStmtAttrW CLI function 5
- SQLSetStmtOption deprecated CLI function 293
- SQLSpecialColumns CLI function
  - details 294
  - Unicode version 5
- SQLSpecialColumnsW CLI function 5
- SQLSTATE
  - format 314
- SQLStatistics CLI function
  - details 299
  - Unicode version 5
- SQLStatisticsW CLI function 5
- SQLTablePrivileges CLI function
  - details 304
  - Unicode version 5
- SQLTablePrivilegesW CLI function 5
- SQLTables CLI function
  - details 307
  - Unicode version 5
- SQLTablesW CLI function 5
- SQLTransact deprecated CLI function 312
- SSLClientKeystash configuration parameter
  - details 400
- SSLClientKeystoredb configuration parameter
  - details 400
- SSLClientKeystoreDBPassword configuration parameter
  - details 401
- SSLClientLabel configuration parameter
  - details 399
- statement attributes
  - CLI
    - changing 427
    - getting 216
    - list 466
    - setting 289
- statement handles
  - allocating 7
  - freeing 146
- StaticCapFile CLI/ODBC configuration keyword 401
- StaticLogFile CLI/ODBC configuration keyword 402
- StaticMode CLI/ODBC configuration keyword 402
- StaticPackage CLI/ODBC configuration keyword 402
- statistics
  - CLI function 299
  - getting 299
- StmtConcentrator CLI/ODBC configuration keyword 403
- StreamGetData CLI/ODBC configuration keyword 403
- StreamPutData CLI/ODBC configuration keyword 404
- strings
  - obtaining start position 212
- SysSchema CLI/ODBC configuration keyword 404

## T

- table privileges CLI function 304
- tables
  - getting table information by using CLI function 307
- TableType CLI/ODBC configuration keyword 405
- target database partition servers
  - logical nodes 341
- TargetPrincipal CLI/ODBC configuration keyword 406
- TempDir CLI/ODBC configuration keyword 406

- terms and conditions
  - publications 548
- TIME data types
  - conversion to C 520
  - display size 536
  - length 535
  - precision 532
  - scale 534
- TIMESTAMP data type
  - conversion to C 520
  - display size 536
  - length 535
  - precision 532
  - scale 534
- TimestampTruncErrToWarning CLI/ODBC configuration
  - keyword 407
- Trace CLI/ODBC configuration keyword 407
- TraceAPIList CLI/ODBC configuration keyword 408
- TraceAPIList! CLI/ODBC configuration keyword 410
- TraceComm CLI/ODBC configuration keyword 412
- TraceErrImmediate CLI/ODBC configuration keyword 413
- TraceFileName CLI/ODBC configuration keyword 413
- TraceFlush CLI/ODBC configuration keyword 414
- TraceFlushOnError CLI/ODBC configuration keyword 415
- TraceLocks CLI/ODBC configuration keyword 415
- TracePathName CLI/ODBC configuration keyword 417
- TracePIDList CLI/ODBC configuration keyword 416
- TracePIDTID CLI/ODBC configuration keyword 417
- TraceRefreshInterval CLI/ODBC configuration keyword 418
- TraceStmtOnly CLI/ODBC configuration keyword 418
- TraceTime CLI/ODBC configuration keyword 419
- TraceTimestamp CLI/ODBC configuration keyword 419
- transactions
  - ending in CLI 95
- troubleshooting
  - online information 548
  - tutorials 548
- Trusted\_Connection CLI/ODBC configuration keyword 420
- tutorials
  - list 547
  - problem determination 548
  - troubleshooting 548
  - Visual Explain 547
- TxnIsolation CLI/ODBC configuration keyword 421

## U

- UID CLI/ODBC configuration keyword 422
- Underscore CLI/ODBC configuration keyword 422
- Unicode UCS-2 encoding
  - CLI
    - functions 5
- updates
  - DB2 Information Center 545, 546
- UseOldStpCall CLI/ODBC configuration keyword 423
- UseServerMsgSP CLI/ODBC configuration keyword 423

## V

- VARBINARY data type
  - conversion to C 520
  - display size 536
  - length 535
  - precision 532
  - scale 534

VARCHAR data type  
conversion to C 520  
display size 536  
length 535  
precision 532  
scale 534

VARGRAPHIC data type  
conversion to C 520  
display size 536  
length 535  
precision 532  
scale 534

## W

WarningList CLI/ODBC configuration keyword 424

WCHAR SQL data type  
display size 536  
length 535  
precision 532  
scale 534

WLONGVARCHAR SQL data type  
display size 536  
length 535  
precision 532  
scale 534

WVARCHAR SQL data type  
display size 536  
length 535  
precision 532  
scale 534

## X

X/Open CAE 314

XMLDeclaration CLI/ODBC configuration keyword 425







Printed in USA

SC27-2438-03



Spine information:

DB2 for Linux, UNIX, and Windows

**Version 9 Release 7**

**Call Level Interface Guide and Reference, Volume 2**

