

# **INFORMIX<sup>®</sup>-CLI**

## Programmer's Manual

Version 2.5  
August 1996  
Part No. 000-8984

Published by INFORMIX® Press

Informix Software, Inc.  
4100 Bohannon Drive  
Menlo Park, CA 94025

Copyright © 1981-1996 by Informix Software, Inc. or their subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

INFORMIX®; INFORMIX-Online Dynamic Server

The following are worldwide trademarks of the indicated owners or their subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

FTP Software: PC/TCP®  
Information Builders, Inc.: API/SQL™; EDA/SQL™; Enterprise Data Access™; Enterprise Data Access/SQL™  
Microsoft Corporation: Microsoft®; MS-DOS®; MS Windows™; Windows™; Windows NT™; Windows® 95;  
Windows for Workgroups™; ODBC™  
Sun Microsystems Inc.: Solaris®  
X/Open Company Ltd.: UNIX®; X/Open®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

Portions of this software product include or were based on INTERSOLV DataDirect ODBC Drivers, a product of INTERSOLV, Inc. Portions copyright INTERSOLV, Inc., 1995. DataDirect is a trademark of INTERSOLV, Inc., 9420 Key West Avenue, Rockville, MD, 20850.

This book incorporates text that is copyright 1994 Microsoft Corporation. This text was taken by permission from Microsoft's *Programmer's Reference: Microsoft Open Database Software Development Kit*, Version 2.0.

Documentation Team: Twila Booth, Diana Chase, Geeta Karmarkar, Katherine Schaefer

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

#### RESTRICTED RIGHTS/SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

---

# Table of Contents

## Introduction

About This Manual . . . . .	3
Organization of This Manual . . . . .	4
Types of Users . . . . .	5
Software Dependencies . . . . .	6
Demonstration Database . . . . .	6
New Features of This Product . . . . .	7
Conventions . . . . .	8
Typographical Conventions . . . . .	8
Icon Conventions . . . . .	9
Example Code Conventions . . . . .	11
Screen-Illustration Conventions . . . . .	11
Additional Documentation . . . . .	12
Printed Documentation . . . . .	12
On-Line Documentation . . . . .	13
Related Reading . . . . .	14
Compliance with Industry Standards . . . . .	15
Informix Welcomes Your Comments . . . . .	15

## Chapter 1

### Overview of INFORMIX-CLI

What Is INFORMIX-CLI? . . . . .	1-3
Advantages of Using INFORMIX-CLI . . . . .	1-5
Overview of the Initialization Files . . . . .	1-6
Understanding the odbcinst.ini File . . . . .	1-6
Understanding the odbc.ini File . . . . .	1-7
File Format for odbc.ini . . . . .	1-8
File Examples for odbc.ini . . . . .	1-11
ODBC Conformance Levels . . . . .	1-12
API Conformance Level of INFORMIX-CLI . . . . .	1-12
SQL Conformance Level of INFORMIX-CLI . . . . .	1-14

	Mapping Data Types . . . . .	1-15
	Supported Isolation and Lock Levels . . . . .	1-16
<b>Chapter 2</b>	<b>INFORMIX-CLI</b>	
	INFORMIX-CLI for UNIX . . . . .	2-3
	Setting Up INFORMIX-CLI . . . . .	2-3
	System Requirements . . . . .	2-4
	Setting Environment Variables . . . . .	2-4
	Adding and Modifying Data Sources . . . . .	2-6
	Adding a Data Source . . . . .	2-6
	Sample Data-Source Entry for .odbc.ini . . . . .	2-10
	Modifying a Data Source . . . . .	2-11
	Connecting to a Data Source . . . . .	2-11
	Using Dialog Boxes to Connect to a Data Source . . . . .	2-11
	Using a Connection-String to Connect to a Data Source . . . . .	2-13
	INFORMIX-CLI for Windows . . . . .	2-15
	Setting Up INFORMIX-CLI . . . . .	2-15
	System Requirements . . . . .	2-15
	Setting the INFORMIXDIR Environment Variable . . . . .	2-16
	Adding and Modifying Data Sources . . . . .	2-16
	Adding a Data Source . . . . .	2-16
	Modifying a Data Source . . . . .	2-24
	Connecting to a Data Source . . . . .	2-26
	Using Dialog Boxes to Connect to a Data Source . . . . .	2-26
	Using a Connection-String to Connect to a Data Source . . . . .	2-28
<b>Chapter 3</b>	<b>Guidelines for Calling INFORMIX-CLI Functions</b>	
	Determining INFORMIX-CLI Conformance Levels . . . . .	3-3
	Using the Driver Manager . . . . .	3-4
	Calling Functions . . . . .	3-5
	Buffers . . . . .	3-5
	Environment, Connection, and Statement Handles . . . . .	3-7
	Using Data Types. . . . .	3-8
	Function Return Codes. . . . .	3-8
<b>Chapter 4</b>	<b>Basic Application Steps</b>	
	How an Application Uses the INFORMIX-CLI Interface . . . . .	4-3

<b>Chapter 5</b>	<b>Connecting to a Data Source</b>	
	Description of Data Sources . . . . .	5-3
	Initializing the Environment . . . . .	5-4
	Allocating a Connection Handle . . . . .	5-5
	Connecting to a Data Source . . . . .	5-5
	Connecting with SQLConnect . . . . .	5-5
	Connecting to a Data Source with SQLDriverConnect . . . . .	5-6
	Connection Browsing with SQLBrowseConnect . . . . .	5-9
	Translating Data . . . . .	5-9
	Related Functions . . . . .	5-10
<b>Chapter 6</b>	<b>Executing SQL Statements</b>	
	Allocating a Statement Handle . . . . .	6-5
	Executing an SQL Statement . . . . .	6-5
	Prepared Execution . . . . .	6-6
	Direct Execution . . . . .	6-7
	Setting Parameter Values . . . . .	6-7
	Performing Transactions . . . . .	6-9
	Retrieving Information About the Data-Source Catalog . . . . .	6-9
	Sending Parameter Data at Execution Time . . . . .	6-10
	Specifying Arrays of Parameter Values . . . . .	6-11
	Using ODBC Extensions to SQL . . . . .	6-12
	Related Functions . . . . .	6-19
<b>Chapter 7</b>	<b>Retrieving Results</b>	
	Assigning Storage for Results (Binding) . . . . .	7-4
	Determining the Characteristics of a Result Set . . . . .	7-4
	Fetching Result Data . . . . .	7-5
	Using Cursors . . . . .	7-6
	Retrieving Data from Unbound Columns . . . . .	7-7
	Assigning Storage for Rowsets (Binding) . . . . .	7-7
	Retrieving Rowset Data . . . . .	7-9
	Using Block and Scrollable Cursors . . . . .	7-9
	Modifying Result Set Data with Positioned Update and Delete Statements . . . . .	7-13
	Processing Multiple Results . . . . .	7-15

<b>Chapter 8</b>	<b>Retrieving Status and Error Information</b>	
	Function Return Codes . . . . .	8-3
	Retrieving Error Messages . . . . .	8-4
	Error Messages . . . . .	8-5
	Error Text Format . . . . .	8-5
	Sample Error Messages . . . . .	8-6
	Processing Error Messages . . . . .	8-8
<b>Chapter 9</b>	<b>Terminating Transactions and Connections</b>	
	Terminating Statement Processing . . . . .	9-3
	Terminating Transactions . . . . .	9-4
	Terminating Connections . . . . .	9-4
<b>Chapter 10</b>	<b>Constructing an INFORMIX-CLI Application</b>	
	Static SQL Example . . . . .	10-4
	Interactive Ad-Hoc Query Example . . . . .	10-7
<b>Chapter 11</b>	<b>Designing Performance-Oriented Applications</b>	
	Connecting to a Data Source . . . . .	11-4
	Executing Calls . . . . .	11-11
	SQLPrepare/SQLExecute Versus SQLExecDirect. . . . .	11-11
	Committing Data . . . . .	11-16
<b>Chapter 12</b>	<b>Function Summary</b>	
	INFORMIX-CLI Function Summary . . . . .	12-3
	Connecting to a Data Source . . . . .	12-4
	Obtaining Information About a Driver and Data Source . . . . .	12-4
	Setting and Retrieving Driver Options . . . . .	12-5
	Preparing SQL Requests . . . . .	12-5
	Submitting SQL Requests . . . . .	12-6
	Retrieving Results and Information about Results . . . . .	12-7
	Obtaining Information About Data-Source System Tables. . . . .	12-8
	Terminating a Statement . . . . .	12-9
	Terminating a Connection. . . . .	12-9
	Setup Shared-Library Function Summary . . . . .	12-10
	Translation Shared-Library Function Summary . . . . .	12-10

<b>Chapter 13</b>	<b>INFORMIX-CLI Function Reference</b>	
	Arguments . . . . .	13-4
	INFORMIX-CLI Include Files . . . . .	13-7
	Diagnostics . . . . .	13-7
	Tables and Views . . . . .	13-7
	Catalog Functions . . . . .	13-8
	Search Pattern Arguments . . . . .	13-8
	SQLColAttributes . . . . .	13-52
	SQLColumnPrivileges . . . . .	13-60
	Related Functions . . . . .	13-90
	SQLDriverConnect . . . . .	13-93
	SQLFreeConnect . . . . .	13-150
	SQLMoreResults . . . . .	13-225
	SQLParamOptions . . . . .	13-239
	SQLProcedures . . . . .	13-255
	SQLSpecialColumns . . . . .	13-295
	SQLTablePrivileges . . . . .	13-312
<b>Chapter 14</b>	<b>Setup Shared Library Function Reference</b>	
	ConfigDSN . . . . .	14-3
	ConfigTranslator . . . . .	14-7
<b>Chapter 15</b>	<b>Translation Shared Library Function Reference</b>	
	SQLDataSourceToDriver . . . . .	15-3
	SQLDriverToDataSource . . . . .	15-7
<b>Appendix A</b>	<b>INFORMIX-CLI Error Codes</b>	
<b>Appendix B</b>	<b>SQL Grammar</b>	
<b>Appendix C</b>	<b>Data Types</b>	
<b>Appendix D</b>	<b>Comparison Between INFORMIX-CLI and Embedded SQL</b>	
	<b>Index</b>	





---

# Introduction

About This Manual . . . . .	3
Organization of This Manual . . . . .	4
Types of Users . . . . .	5
Software Dependencies . . . . .	6
Demonstration Database . . . . .	6
New Features of This Product . . . . .	7
Conventions . . . . .	8
Typographical Conventions . . . . .	8
Icon Conventions . . . . .	9
Comment Icons . . . . .	9
Compliance Icons . . . . .	10
Platform Icons . . . . .	10
Example Code Conventions . . . . .	11
Screen-Illustration Conventions . . . . .	11
Additional Documentation . . . . .	12
Printed Documentation . . . . .	12
On-Line Documentation. . . . .	13
Error Message Files . . . . .	13
Release Notes and Documentation Notes . . . . .	14
Related Reading . . . . .	14
Compliance with Industry Standards . . . . .	15
Informix Welcomes Your Comments . . . . .	15



**T**his chapter introduces the *INFORMIX-CLI Programmer's Manual*. Read this chapter for an overview of the information provided in this manual and for an understanding of the conventions used throughout this manual.

The INFORMIX-CLI application-programming interface (API) enables the C programmer to create custom applications for relational database access, and it uses function calls to pass dynamic SQL statements as function arguments. It is an alternative to embedded dynamic SQL. Unlike embedded SQL, it does not require a preprocessor.

The libraries and header files that are provided with INFORMIX-CLI enable you to access databases, manipulate the data in your program, interact with the database server, and check for errors.

---

## About This Manual

The *INFORMIX-CLI Programmer's Manual* is both a user guide and a reference manual. This manual is a complete guide to the features that make up the Informix implementation of the ODBC standards. This manual explains how to use INFORMIX-CLI to access an Informix database server and describes the special features that the product offers. It progresses from general topics to more advanced programming techniques and examples.

## Organization of This Manual

The *INFORMIX-CLI Programmer's Manual* includes the following chapters:

- This Introduction provides an overview of the manual and describes the documentation conventions used.
- [Chapter 1, “Overview of INFORMIX-CLI,”](#) introduces INFORMIX-CLI, describes its relationship to ODBC, describes the format of the initialization files, and lists the API and SQL grammar conformance levels.
- [Chapter 2, “INFORMIX-CLI,”](#) explains how to set up the INFORMIX-CLI driver and how to configure and connect to data sources for UNIX and Windows platforms.
- [Chapter 3, “Guidelines for Calling INFORMIX-CLI Functions,”](#) describes the role of the driver manager and the general characteristics of INFORMIX-CLI functions.
- [Chapter 4, “Basic Application Steps,”](#) describes how an application uses the INFORMIX-CLI interface to interact with a data source.
- [Chapter 5, “Connecting to a Data Source,”](#) describes data sources and also describes how to establish a connection to a data source.
- [Chapter 6, “Executing SQL Statements,”](#) describes the sequence of function calls INFORMIX-CLI applications can use to execute SQL statements.
- [Chapter 7, “Retrieving Results,”](#) describes the SQL statements that return and retrieve results.
- [Chapter 8, “Retrieving Status and Error Information,”](#) defines the INFORMIX-CLI return codes and error-handling protocol.
- [Chapter 9, “Terminating Transactions and Connections,”](#) describes how an application terminates statements and transactions. The chapter also discusses how an application terminates the connection when it finishes interacting with a data source.
- [Chapter 10, “Constructing an INFORMIX-CLI Application,”](#) provides code examples of basic applications.

- [Chapter 11, “Designing Performance-Oriented Applications,”](#) provides guidelines for designing and coding efficient INFORMIX-CLI applications.
- [Chapter 12, “Function Summary,”](#) summarizes the functions that INFORMIX-CLI uses.
- [Chapter 13, “INFORMIX-CLI Function Reference,”](#) describes each INFORMIX-CLI function in detail. Each reference contains the function’s ODBC conformance level, purpose, syntax, returns, diagnostics, comments, SQLSTATE values, and related functions. The chapter also includes code examples.
- [Chapter 14, “Setup Shared Library Function Reference,”](#) describes the syntax of the driver-setup and translator-setup shared-library APIs.
- [Chapter 15, “Translation Shared Library Function Reference,”](#) describes the syntax of the translation shared-library API.
- [Appendix A](#) provides a listing of the SQLSTATE values that can be returned, the error message that corresponds to each numerical value (SQLSTATE), and the functions that can return each SQLSTATE value.
- [Appendix B](#) describes the recommended syntax for maximum interoperability in calls to **SQLPrepare**, **SQLExecute**, and **SQLExecDirect**.
- [Appendix C](#) provides information on ODBC SQL data types that INFORMIX-CLI supports and ODBC C data types. This appendix also describes conversion from one data type to the other.
- [Appendix D](#) compares INFORMIX-CLI functions (core ODBC) with embedded SQL statements.

## Types of Users

This manual is for C programmers who want to use INFORMIX-CLI to create custom applications for accessing Informix relational databases. The manual assumes that you know C programming and are familiar with your computer operating system and the structure of relational databases. Knowledge of SQL is also useful. Three other manuals describe the Informix implementation of SQL in detail: the [Informix Guide to SQL: Tutorial](#), the [Informix Guide to SQL: Reference](#), and the [Informix Guide to SQL: Syntax](#).

## Software Dependencies

Depending on your Informix database configuration, you must have a compatible Informix database server installed on your system or network. The following list shows examples of compatible database servers:

- INFORMIX-OnLine (Version 5.x)
- INFORMIX-OnLine Dynamic Server (Version 7.x)
- INFORMIX-SE (Version 5x, 7.x)

For information on these products, refer to your Informix documentation set. For a complete listing of compatible database servers, see the release notes for INFORMIX-CLI.

INFORMIX-CLI enables you to compose queries, send them to the database server, and view the results that the database server returns. You can use DB-Access to test and verify all the SQL statements described in this manual.

## Demonstration Database

The DB-Access utility, which is provided with your Informix database server products, includes a demonstration database that contains information about a fictitious wholesale sporting-goods distributor. The sample command files that make up a demonstration application are also included.

Most examples in this manual are based on the **stores7** demonstration database. The **stores7** database is described in detail and its contents are listed in Appendix A of the [Informix Guide to SQL: Reference](#).

For the script and the instructions that you use to install the demonstration database on your database server, refer to your Informix database server documentation.

---

## **New Features of This Product**

This section highlights the major new features implemented in Version 2.5 of INFORMIX-CLI:

- **Support for Windows 95 and Windows NT**  
INFORMIX-CLI now supports application programming for both Windows NT and Windows 95 environments.
- **Global Language Support (GLS)**  
The GLS feature lets Informix Version 7.2 products handle different languages, cultural conventions, and code sets. GLS functionality supersedes the functionality of Native Language Support (NLS) and Asian Language Support (ALS). GLS eliminates the need to distinguish between internationalized versions of Informix software.  
This feature is available only when you connect to an Informix database server of Version 7.2 or later.
- **Support for additional ODBC API level 2 functions:**
  - SQLMoreResults
  - SQLColumnPrivileges
  - SQLTablePrivileges
  - SQLParamOptions
  - SQLProcedures

---

## Conventions

This section describes the conventions that are used in this manual. By becoming familiar with these conventions, you will find it easier to gather information from this and other volumes in the documentation set.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Example-code conventions
- Screen-illustration conventions

## Typographical Conventions

This manual uses a standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth. The following typographical conventions are used throughout this manual.

---

Convention	Meaning
<i>italics</i>	Within text, new terms and emphasized words are printed in italics. Within syntax diagrams, values that you are to specify are printed in italics. Within examples, values that you must supply, such as a database, file, or program name are printed in italics. A table following the diagram explains the value.
<b>boldface</b>	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, table names, column names, menu items, command names, and other similar terms are printed in boldface.
<code>monospace</code>	Information that the product displays and information that you enter are printed in a monospace typeface.
KEYWORD	All keywords appear in uppercase letters.
◆	This symbol indicates the end of product- or platform-specific information.

---








**Tip:** When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

## Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

### Comment Icons

Comment icons identify three types of information, as described in the following table. This information is always displayed in italics.

Icon	Description
	Identifies paragraphs that contain vital instructions, cautions, or critical information.
	Identifies paragraphs that contain significant information about the feature or operation that is being described.
	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described.

### ***Compliance Icons***

Compliance icons indicate paragraphs that provide guidelines for complying with a standard.

<b>Icon</b>	<b>Description</b>
<b>GLS</b>	Identifies information that is specific to a database or application that uses the Informix Global Language Support (GLS) feature for support of a nondefault locale.
<b>Core</b>	Identifies that a function supports the Core ODBC API conformance level.
<b>Level 1</b>	Identifies that a function supports the Level 1 ODBC API conformance level.
<b>Level 2</b>	Identifies that a function supports the Level 2 ODBC API conformance level.

### ***Platform Icons***

Platform icons identify paragraphs that describe product-specific or platform-specific information. The following table describes the product and platform icons that are used in this manual.

<b>Icon</b>	<b>Description</b>
<b>UNIX</b>	Identifies information that is specific to 32-bit support for the UNIX environment.
<b>Windows 3.1</b>	Identifies information that is specific to 16-bit support for the Windows 3.1x environment.
<b>Windows NT</b>	Identifies information that is specific to the Windows NT environment.
<b>Windows 95</b>	Identifies information that is specific to the Windows 95 environment.

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the product- or platform-specific information.

## **Example Code Conventions**

Examples of C and SQL code occur throughout this manual. The code is not specific to any single Informix application development tool, except where it is noted otherwise.

Note that an ellipsis (...) in example code indicates that arguments can be repeated several times. Also note that a column of three dots in an example indicates that additional code would be added in a full application, but it is not necessary to show more code to describe the concept being discussed.

## **Screen-Illustration Conventions**

The illustrations in this manual represent a generic rendition of various windowing environments. The details of dialog boxes, controls, and windows were deleted or redesigned to provide this generic look. Therefore, the illustrations in this manual depict the windowing environment a little differently than the way it appears on your screen.

---

## Additional Documentation

This section describes the following pieces of the documentation set:

- Printed documentation
- On-line documentation
- Related reading

### Printed Documentation

The following related Informix documents complement the information in this manual:

- You, or whoever installs your Informix products, should refer to the [INFORMIX-CLI Installation Guide](#) to ensure that your Informix product is properly set up before you begin to work with it.
- If you have never used Structured Query Language (SQL), read the [Informix Guide to SQL: Tutorial](#). It provides a tutorial on Informix SQL. It also describes the fundamental ideas and terminology for planning, implementing, and using a relational database.
- A companion volume to the Tutorial, the [Informix Guide to SQL: Reference](#), includes details of the Informix system catalog tables, describes Informix environment variables and UNIX environment variables that you should set, and describes the column data types that Informix database servers support.
- An additional companion volume to the Reference, the [Informix Guide to SQL: Syntax](#), provides a detailed description of all the SQL statements supported by Informix products. This guide also provides a detailed description of Stored Procedure Language (SPL) statements.
- The [SQL Quick Syntax Guide](#) contains syntax diagrams for all statements and segments described in this manual.
- Depending on the database server you are using, you or your system administrator need either the [INFORMIX-SE Administrator's Guide](#) or the [INFORMIX-OnLine Dynamic Server Administrator's Guide](#).

- The [Guide to GLS Functionality](#) describes how to use nondefault locales with Informix products. A locale contains language- and culture-specific information that Informix products use when they format and interpret data.
- The [DB-Access User Manual](#) describes how to invoke the DB-Access utility to access, modify, and retrieve information from Informix database servers.
- When errors occur, you can look them up by number and learn their cause and solution in the [Informix Error Messages](#) manual. If you prefer, you can look up the error messages in the on-line message file that is described in “[Error Message Files](#)” later in this Introduction and in the Introduction to the [Informix Error Messages](#) manual.

## On-Line Documentation

Several different types of on-line documentation are available:

- On-line error messages
- Release notes and documentation notes

### *Error Message Files*

Informix software products provide ASCII files that contain all of the Informix error messages and their corrective actions. To read the error messages in the ASCII file, Informix provides scripts that let you display error messages on the screen (**finderr**) or print formatted error messages (**rofferr**). See the Introduction to the [Informix Error Messages](#) manual for a detailed description of these scripts.

## Release Notes and Documentation Notes

In addition to the Informix set of manuals, the following on-line files, located in the `$INFORMIXDIR/release/en_us/0333` directory, might supplement the information in this manual.

---

On-Line File	Purpose
Documentation notes	Describes features that are not covered in the manuals or that have been modified since publication. The file that contains the documentation notes for INFORMIX-CLI, Version 2.5 for UNIX is called <code>CLIDOC_2.5</code> . The file that contains the documentation notes for INFORMIX-CLI, Version 2.5 for Windows is called <code>doccli.250</code> .
Release notes	Describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds. The file that contains the release notes for INFORMIX-CLI, Version 2.5 for UNIX is called <code>CLI_2.5</code> . The file that contains the release notes for INFORMIX-CLI, Version 2.5 for Windows is called <code>cli.250</code> .

---

Please examine these files because they contain vital information about application and performance issues.

## Related Reading

For additional information on ODBC, refer to the *Microsoft ODBC Programmer's Reference and SDK Guide*, Version 2.0 (Microsoft Press, 1994).

For additional technical information on database management, consult the following books.

- *An Introduction to Database Systems* by C. J. Date (Addison-Wesley Publishing, 1994)
- *Transaction Processing: Concepts and Techniques* by Jim Gray and Andreas Reuter (Morgan Kaufmann Publishers, Inc., 1993)

To learn more about the SQL language, consider the following books:

- *A Guide to the SQL Standard* by C. J. Date with H. Darwen (Addison-Wesley Publishing, 1993)
- *Understanding the New SQL: A Complete Guide* by J. Melton and A. Simon (Morgan Kaufmann Publishers, 1993)
- *Using SQL* by J. Groff and P. Weinberg (Osborne McGraw-Hill, 1990)

---

## Compliance with Industry Standards

INFORMIX-CLI is based on the Microsoft Open Database Connectivity (ODBC) Specification, which in turn is based on the X/Open and SQL Access Group Call Level Interface (CLI) Specification. The ODBC and CLI specifications provide a common and open interface through which ANSI-compliant SQL is passed.

---

## Informix Welcomes Your Comments

Please let us know what you like or dislike about our manuals. To help us with future versions of our manuals, please tell us about any corrections or clarifications that you would find useful. Please include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.  
SCT Technical Publications Department  
4100 Bohannon Drive  
Menlo Park, CA 94025

If you prefer to send electronic mail, our address is:

**doc@informix.com**

Or send a facsimile to the Informix Technical Publications Department at:

**415-926-6571**

We appreciate your feedback.



---

# Overview of INFORMIX-CLI

What Is INFORMIX-CLI? . . . . .	1-3
Advantages of Using INFORMIX-CLI . . . . .	1-5
Overview of the Initialization Files . . . . .	1-6
Understanding the odbcinstr.ini File . . . . .	1-6
Understanding the odbc.ini File . . . . .	1-7
File Format for odbc.ini . . . . .	1-8
ODBC Data Sources . . . . .	1-8
Data-Source Specification . . . . .	1-9
Default Data Source Specification . . . . .	1-10
ODBC Options . . . . .	1-10
File Examples for odbc.ini . . . . .	1-11
ODBC Conformance Levels . . . . .	1-12
API Conformance Level of INFORMIX-CLI . . . . .	1-12
SQL Conformance Level of INFORMIX-CLI . . . . .	1-14
Mapping Data Types . . . . .	1-15
Supported Isolation and Lock Levels. . . . .	1-16



# T

his chapter introduces INFORMIX-CLI and describes its relationship to ODBC, describes the format of the initialization files, lists the API and SQL grammar conformance levels, and describes the error message format.

---

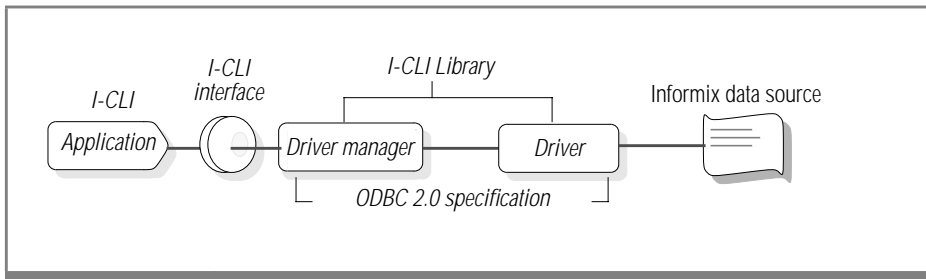
## What Is INFORMIX-CLI?

INFORMIX-CLI is a call-level interface (CLI) for SQL that provides a library of C-language function calls that support SQL statements. INFORMIX-CLI is based on the Microsoft Open Database Connectivity (ODBC) 2.5 Specification. INFORMIX-CLI applications enable ODBC functionality through the use of the INFORMIX-CLI library.

The member functions and constants defined in the INFORMIX-CLI library correspond to specific ODBC functions and constants, except where noted otherwise in this manual.

The INFORMIX-CLI library includes two components: the driver manager and the driver. The driver manager provides information to an application, loads drivers dynamically, and provides argument and transition checking. The driver is either a shared library (for UNIX drivers) or a dynamic link library (for Windows drivers) that implements function calls and manages exchanges between an application and an Informix database.

Figure 1-1 shows the relationship of the components in an INFORMIX-CLI application and ODBC configuration. The driver manager and driver appear to an application as one unit that processes INFORMIX-CLI function calls.



**Figure 1-1**  
INFORMIX-CLI  
Architecture

The *data source* consists of the database a user wants to access, its associated DBMS, the platform on which the DBMS resides, and the network used to access that platform.

When you compile and link an INFORMIX-CLI application, it is automatically equipped to communicate with an Informix database server that resides either on the same computer (*local*) or on a network on other computers (*remote*).

INFORMIX-CLI enables the development of applications that issue SQL statements to Informix database servers and process resulting data dynamically. INFORMIX-CLI provides functions that allow your applications to perform the following set of operations:

- Initiate and terminate connections to database servers
- Obtain information about the server and CLI
- Set and retrieve CLI options
- Prepare and submit SQL requests
- Retrieve results and information about results
- Obtain information about server system tables

---

## **Advantages of Using INFORMIX-CLI**

Based on the Microsoft Open Database Connectivity (ODBC) Specification, a call level interface such as INFORMIX-CLI is typically used for dynamic access. INFORMIX-CLI is similar to the dynamic embedded version of SQL described in the X/Open and SQL Access Group CAE specification (1992).

Applications written with INFORMIX-CLI do not need embedded Structured Query Language (SQL) statements. Instead, they pass SQL statement strings to INFORMIX-CLI functions as arguments.

For a comparison between embedded SQL statements and the ODBC call-level interface on which INFORMIX-CLI is based, see [Appendix D](#).

INFORMIX-CLI is very straightforward to application developers who are familiar with function libraries. The function-call interface does not require host variables or other embedded SQL concepts.

INFORMIX-CLI does not require a preprocessor. To submit an SQL request, you place an SQL command into a text buffer and pass the buffer as a parameter in a function call. INFORMIX-CLI functions provide declarative capabilities and request management. You obtain error information as you do for any function call, by return code or error function call, depending on the application.

INFORMIX-CLI allows for specification of result storage before or after the results are available. Results can be determined and appropriate action taken without being limited to a specific set of data structures that were defined prior to the request. Deferral of storage specification is called late binding of variables.

---

## Overview of the Initialization Files

The **odbcinst.ini** and **odbc.ini** initialization files are created when you install INFORMIX-CLI. The driver manager uses the **odbcinst.ini** file to determine which drivers and translators are currently installed. The **odbc.ini** file is a template file that contains your data-source configuration information. The following sections describe the format and structure of these initialization files.

---

## Understanding the odbcinst.ini File

The **odbcinst.ini** file contains the following sections:

ODBC Drivers	This section lists the description the Informix driver.
Driver Specification	This section lists the driver shared library and attribute keywords of the Informix driver.
Default Driver Specification	This section specifies the default driver to be used when none is specified. The default driver is the Informix driver.
ODBC Translators	This section lists the description of each available translator.
Translator Specification	Each translator described in the ODBC Translators section has a section that lists the translator shared library

## Understanding the `odbc.ini` File

The INFORMIX-CLI driver and driver manager use the **`odbc.ini`** initialization file. Although this file has a slightly different name depending upon whether you use a UNIX or Windows client application, the format and structure of the file are the same.

### UNIX

For UNIX users, **`odbc.ini`** is a text file called **`.odbc.ini`**. The file is located in your home directory. Initially, a template file called **`.odbc.ini`** resides in the **`SINFORMIXDIR/cli`** directory where your INFORMIX-CLI software is installed. For information on setting up your **`.odbc.ini`** file, see [“Adding and Modifying Data Sources” on page 2-6](#). ♦

### Windows 3.1

For Windows 3.1, **`odbc.ini`** is a text file called **`ODBC.INI`**. The file is located in the **`WINDOWS`** directory. Before you can connect to a data source, you must use the ODBC Administrator to add the connection information for the data source to this file.



***Warning:*** *Windows 3.1 users should never modify the `ODBC.INI` file. The contents of this file are changed based on the data source setup and modification you make using the ODBC Administrator. Modifying the `ODBC.INI` file directly might result in data-source configuration errors.* ♦

### Windows NT

For Windows NT and Windows 95, **`ODBC.INI`** is a subkey of the **`HKEY_CURRENT_USER`** key within the registry. When you access the registry using this subkey, the ODBC structure is the same as described in [“File Format for `odbc.ini`” on page 1-8](#). Before you can connect to a data source, you must use the ODBC Administrator to add the connection information for the data source to the **`ODBC.INI`** subkey. ♦

### Windows 95

## File Format for *odbc.ini*

The **odbc.ini** file includes the following sections:

ODBC DataSources	This section lists the name of each data source and describes its associated driver.
Data Source Specification	Each data source listed in the ODBC Data Sources section has a section that contains additional information about that data source.
Default Data Source Specification	This section is optional and specifies the default data source to use when none is specified.
ODBC Options	This section specifies the ODBC installation directory and the ODBC options that can be enabled or disabled.

### *ODBC Data Sources*

Each entry in the ODBC Data Sources section lists a data source and a description of the driver it uses. This section has the following format:

```
[ODBC Data Sources]
data_source_name1=driver_description1
data_source_name2=driver_description2
.
.
.
```

*data\_source\_name* identifies the data source that the driver accesses. You choose the data-source name.

*driver\_description* describes the database driver that accesses the data source. This field is optional.

For example, to define an **Agencies** data source that uses the INFORMIX-CLI driver, the **odbc.ini** entry might look like the following example:

```
[ODBC Data Sources]
Agencies=Informix Driver
```



## ***Data-Source Specification***

Each data source listed in the ODBC Data Sources section has its own Data Source Specification section. This section has the following format:

```
[data_source_name]
Driver=driver_path
Description=data_source_description
TranslationDLL=translation_path
TranslationName=translator_name
TranslationOption=translation_option
keyword=attribute
.
.
.
```

<i>data_source_name</i>	is the name of the data source, as specified in the ODBC Data Sources section of this file.
<i>driver_path</i>	is the full path to the driver shared library (UNIX) or dynamic link library (Windows).
<i>translation_path</i>	is the full path of the translation shared library (UNIX) or dynamic link library (Windows). This field is optional.
<i>translation_option</i>	shows the ASCII representation of the 32-bit integer translation option. This field is optional.
<i>attribute</i>	specifies the value for the keyword. Each INFORMIX-CLI driver has its own set of keywords. For UNIX-specific keywords and attributes, refer to <a href="#">“Adding a Data Source” on page 2-6</a> . For Windows-specific keywords and attributes, refer to <a href="#">“Adding a Data Source” on page 2-16</a> . This field is optional.

For example, the data source called **Agencies** connects to an INFORMIX-CLI driver for Solaris called **libifmx07.so.1**. The database that **Agencies** accesses is called **agencies**, and it resides on the database server. The Data Source Specification entry for the **Agencies** data source might look like the following example:

```
[Agencies]
Driver=/usr/informix/cli/dlls/qeinf708.so
Database=agencies
LogonID=marvin
```

### ***Default Data Source Specification***

This section (for both UNIX and Windows) is optional. The Default Data Source Specification contains information about the default data source. This data source is called Default and has the same format as any other Data Source Specification section; however, the Default data source is not listed in the ODBC Data Sources section.

The following example shows a Default Data Source Specification entry for an Informix database:

```
[Default]
Driver=/usr/informix/cli/dlls/qeinf708.so
Database=Stores7
LogonID=marvin
```

### ***ODBC Options***

The ODBC Options section indicates whether tracing is enabled or disabled. With tracing, all ODBC function calls made from an application can be logged to the specified trace file. This section has the following format:

```
[ODBC]
Trace=0|1
TraceFile=tracefile_path
TraceAutoStop=0|1
```

This section lists the following information:

- |                  |  |
|------------------|--|
| <b>Trace</b>     | indicates whether tracing is enabled. If the Trace keyword is set to 0, tracing is disabled. If the Trace keyword is set to 1, tracing is enabled. |
| <b>TraceFile</b> | is the name of the specified trace file that is logging the ODBC function calls.   |

<i>tracefile_path</i>	specifies the full path to the trace file. If a trace file is not specified and tracing is enabled, logging information is written to the <b>sql.log</b> file, which is located in your current directory.
TraceAutoStop	indicates whether tracing is enabled or disabled when an application calls the <b>SQLFreeEnv</b> function. If the TraceAutoStop keyword is set to 0, tracing is not automatically disabled. If the TraceAutoStop keyword is set to 1, the Trace keyword in the <b>odbc.ini</b> file is set to 0. Additionally, all other concurrent Windows ODBC applications will have tracing disabled. The default is 1.

## File Examples for *odbc.ini*

The following example shows a Windows 3.1 ODBC.INI file:

```
[ODBC Data Sources]
Stores7=Informix Driver

[Default]
Driver=C:\WINDOWS\SYSTEM\QEINF509.DLL
Database=vendors
LogonID=mary

[Stores7]
Driver=C:\WINDOWS\SYSTEM\QEINF509.DLL
HostName=odin
Service=sqlexec
Database=stores7
LogonID=mary

[ODBC]
Trace=1
TraceFile=C:\WINDOWS\LOG\TRACE.LOG
TraceAutoStop=0
◆
```

Windows

**UNIX**

The following example shows a UNIX **.odbc.ini** file:

```
[ODBC Data Sources]
Stores7=Informix Driver

[Default]
Driver=/usr/informix/cli/dlls/qeinf708.so
Database=stores7
LogonID=marvin

[ODBC]
InstallDir=/usr/informix/cli
Trace=1
TraceFile=~/.trace.log
TraceAutoStop=0
◆
```

---

## ODBC Conformance Levels

ODBC defines two conformance standards for drivers: the API conformance standard and the SQL conformance standard. API conformance refers to the functions that a driver supports. SQL conformance refers to the SQL grammar that the driver supports.

### API Conformance Level of INFORMIX-CLI

The ODBC API conformance standard includes three levels: Core, Level 1, and Level 2. The Core level includes functions that correspond to the functions in the X/Open and SQL Access Group Call Level Interface (CLI) specification. Level 1 and Level 2 functions extend the core functionality.

The following table lists all Core, Level 1, and Level 2 API functions that are defined by the ODBC specification. INFORMIX-CLI supports all of the Core and Level 1 API functions. In addition, INFORMIX-CLI supports all Level 2 functions except those marked with an asterisk (\*).

Core Level Functions	Level 1 Functions	Level 2 Functions
SQLAllocConnect	SQLBindParameter	SQLBrowseConnect
SQLAllocEnv	SQLColumns	SQLColumnPrivileges
SQLAllocStmt	SQLDriverConnect	SQLDataSources
SQLBindCol	SQLGetConnectOption	SQLDescribeParam*
SQLCancel	SQLGetData	SQLDrivers
SQLColAttributes	SQLGetInfo	SQLExtendedFetch
SQLConnect	SQLGetFunctions	SQLForeignKeys*
SQLDescribeCol	SQLGetStmtOption	SQLMoreResults
SQLDisconnect	SQLGetTypeInfo	SQLNativeSql
SQLError	SQLParamData	SQLNumParams
SQLExecDirect	SQLPutData	SQLParamOptions
SQLExecute	SQLSetConnectOption	SQLPrimaryKeys
SQLFetch	SQLSetStmtOption	SQLProcedureColumns*
SQLFreeConnect	SQLSpecialColumns	SQLProcedures
SQLFreeEnv	SQLStatistics	SQLSetPos*
SQLFreeStmt	SQLTables	SQLSetScrollOptions
SQLGetCursorName		SQLTablePrivileges
SQLNumResultCols		
SQLPrepare		

(1 of 2)

Core Level Functions	Level 1 Functions	Level 2 Functions
SQLRowCount		
SQLSetCursorName		
SQLTransact		

(2 of 2)

## SQL Conformance Level of INFORMIX-CLI

The ODBC SQL conformance standard includes three levels: Minimum, Core, and Extended. INFORMIX-CLI supports all Minimum and Core SQL grammar, which includes the following statements, expressions, and data types:

- Data Definition Language (DDL) statements: CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT, and REVOKE
- Data Manipulation Language (DML) statements: full SELECT, INSERT, UPDATE, SEARCHED, and DELETE SEARCHED
- Expressions: simple (such as A>B+C), subquery, set functions such as SUM and MIN
- Data types: CHAR, VARCHAR, LONG VARCHAR, DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, DOUBLE PRECISION

In addition, INFORMIX-CLI supports the following ODBC extensions to SQL grammar:

- DML statements: outer joins, positioned UPDATE, positioned DELETE, SELECT FOR UPDATE, and UNIONS
- Date and time-stamp data
- The following numeric functions:

abs	acos	asin	atan
atan2	cos	cot	exp
log	log10	mod	power
round	sin	sqrt	tan
truncate			

- The following date functions:
 

curdate	dayofweek	now
dayofmonth	month	year
- The following string functions:
 

concat	LTRIM
length	RTRIM
- The user system function
- Data types: LONG VARBINARY, DATE, and TIMESTAMP
- Procedure calls

For more information on the numeric, date, string, and system functions, refer to [Informix Guide to SQL: Syntax](#).

---

## Mapping Data Types

INFORMIX-CLI maps Informix data types to their appropriate ODBC SQL data types. The following table lists the Informix data type and its corresponding ODBC SQL data type.

Informix Data Type	ODBC SQL Data Type
BYTE*	SQL_LONGVARBINARY
CHAR, CHARACTER	SQL_CHAR
DATE	SQL_DATE
DATETIME year to fraction(5)	SQL_TIMESTAMP
DECIMAL, DEC, NUMERIC	SQL_DECIMAL
FLOAT	SQL_DOUBLE
INTEGER, INT	SQL_INTEGER
INTERVAL	SQL_CHAR
MONEY	SQL_DECIMAL

(1 of 2)

Informix Data Type	ODBC SQL Data Type
SERIAL	SQL_INTEGER
SMALLFLOAT, REAL	SQL_REAL
SMALLINT	SQL_SMALLINT
TEXT*	SQL_LONGVARCHAR
VARCHAR,* CHARACTER VARYING*	SQL_VARCHAR

(2 of 2)

\* Not supported for INFORMIX-SE databases

For more information on data types, see [Appendix C](#).

---

## Supported Isolation and Lock Levels

If connected to an OnLine database server, the INFORMIX driver supports isolation levels 0 (Read Uncommitted), 1 (Read Committed), and 3 (Serializable). The default setting is 1. INFORMIX-SE supports isolation level 0 (Read Uncommitted) only.

The Informix driver also supports an alternative isolation level 1, called cursor stability. Your INFORMIX-CLI application can use this isolation level by calling **SQLSetConnectOption** (1040,1).

Additionally, if transaction logging has not been enabled for your database, then transactions are not supported by the driver (the driver is always in auto-commit mode). Each statement is treated as if it is a single transaction.

The Informix driver also supports page-level locking. For more information on isolation and lock levels, see the [Informix Guide to SQL: Tutorial](#).



# INFORMIX-CLI

INFORMIX-CLI for UNIX . . . . .	2-3
Setting Up INFORMIX-CLI . . . . .	2-3
System Requirements . . . . .	2-4
Setting Environment Variables . . . . .	2-4
Adding and Modifying Data Sources . . . . .	2-6
Adding a Data Source . . . . .	2-6
Required Data-Source Configuration Information . . . . .	2-6
Optional Data-Source Configuration Information . . . . .	2-6
Sample Data-Source Entry for .odbc.ini . . . . .	2-10
Modifying a Data Source . . . . .	2-11
Connecting to a Data Source . . . . .	2-11
Using Dialog Boxes to Connect to a Data Source . . . . .	2-11
Using a Connection-String to Connect to a Data Source . . . . .	2-13
INFORMIX-CLI for Windows . . . . .	2-15
Setting Up INFORMIX-CLI . . . . .	2-15
System Requirements . . . . .	2-15
Setting the INFORMIXDIR Environment Variable . . . . .	2-16
Adding and Modifying Data Sources . . . . .	2-16
Adding a Data Source . . . . .	2-16
Required Data-Source Configuration Information . . . . .	2-17
Optional Data-Source Configuration Information . . . . .	2-17
Modifying a Data Source . . . . .	2-24
Connecting to a Data Source . . . . .	2-26
Using Dialog Boxes to Connect to a Data Source . . . . .	2-26
Using a Connection-String to Connect to a Data Source . . . . .	2-28



# T

his chapter explains how to set up INFORMIX-CLI and how to configure and connect to data sources for UNIX and Windows platforms.

---

## INFORMIX-CLI for UNIX

INFORMIX-CLI for UNIX conforms to the *Microsoft ODBC Specification (Version 2.5)*. INFORMIX-CLI supports multiple connections to an Informix database. The INFORMIX-CLI library is named **qainf708**. Some libraries might have a platform-specific extension; for example, the library for Solaris is called **qainf708.so**.

The following sections describe how to set up and configure INFORMIX-CLI so that you can connect your UNIX client application to a data source.

---

## Setting Up INFORMIX-CLI

Perform the following setup activities before you use INFORMIX-CLI:

- Confirm that your system has the appropriate software installed.
- Set the **INFORMIXDIR** and **PATH** environment variables to reflect the appropriate directory paths.
- Set the **INFORMIXSERVER** environment variable to the default database server name

## System Requirements

Depending on your Informix database configuration, you must have a compatible Informix database server installed on your system or network. The following list shows examples of compatible database servers:

- INFORMIX-OnLine (Version 5.x)
- INFORMIX-OnLine Dynamic Server (Version 7.x)
- INFORMIX-SE (Version 5x, 7.x)

For information on these products, refer to your Informix documentation set. For a complete listing of compatible database servers, see the release notes for INFORMIX-CLI.

### GLS

To take advantage of the GLS feature, you must have an Informix database server of Version 7.2 or greater. For more information, see the [Guide to GLS Functionality](#). ♦

## Setting Environment Variables

Set the **INFORMIXDIR** environment variable to the full path of the directory where your Informix product is installed. The **INFORMIXSERVER** environment variable must specify the default database server for the user. This value must correspond to a valid **dbservername** entry in the **sqlhosts** file. The UNIX environment variable, **PATH**, indicates the directories that are searched for executable programs. Your **PATH** setting must include the path to your **\$INFORMIXDIR/bin** directory. In the C shell, you can set these variables using the **setenv** command at the command line or in your **.login** or **.cshrc** files. In the Bourne or Korn shells, you can set this variable using the **export** command at the command line or in your **.login** or **.profile** files. If you set these variables at the command line, you must reset them whenever you log on to your system. If you set these variables in a file, they are set automatically when you log on to your system.

For example, if your Informix directory path is **/usr/informix** and the name of your default database server is **online\_one**, in the C shell you could add the following lines to your **.cshrc** file to set the **INFORMIXDIR**, **INFORMIXSERVER**, and **PATH** environment variables:

```
setenv INFORMIXDIR /usr/informix
setenv INFORMIXSERVER online_one
setenv PATH ${PATH}:${INFORMIXDIR}/bin
```

In the Bourne or Korn shells, you could add the following lines to your **.login** or **.profile** file:

```
INFORMIXDIR=/usr/informix; export INFORMIXDIR
INFORMIXSERVER=online_one; export INFORMIXSERVER
PATH=${PATH}:${INFORMIXDIR}/bin; export PATH
```

## GLS

When you installed **INFORMIX-CLI**, the following environment variables got set:

- **IV\_GLS\_LCDIR**
- **IV\_GLS\_REGISTRY**
- **GL\_PATH**

Look in your **.cli.sh** file (Bourne shell) or **.cli.csh** (C Shell) to see the settings. For information about additional GLS environment variables that you can set, see the [Guide to GLS Functionality](#). ♦

---

## **Adding and Modifying Data Sources**

A *data source* is a database or file that INFORMIX-CLI accesses. INFORMIX-CLI uses the **.odbc.ini** file for initialization. To connect to a data source, the driver manager looks at your **.odbc.ini** file for specific connection information. This file contains information about each data source. UNIX users are responsible for modifying their **.odbc.ini** file using a text editor.

### **Adding a Data Source**

Before you can connect to a data source, you must add an entry for that data source in your **.odbc.ini** file. For complete information on the format and contents of the **.odbc.ini** file, refer to [“Understanding the odbc.ini File” on page 1-7](#).

### ***Required Data-Source Configuration Information***

When you add a data source, you must provide two pieces of information in the Data Source Specification section: the name of the data source and the full path to your driver shared library. All other connection information is optional.

### ***Optional Data-Source Configuration Information***

When you add a data source, you can choose to define new connection defaults. You can specify two types of connection options in the Data-Source Specification section:

- Options that define names
- Options that define cursor behavior

*Options That Define Names*

The following table lists the names that you can set as default connection options for a data source.

Keyword	Attribute
Description	A longname that you choose to identify the data source.
Database	The name of the OnLine or SE database that the data source accesses. The name can include the database server qualifier.
LogonID	Your user name as specified on the INFORMIX-OnLine Dynamic Server, or the INFORMIX-SE database server.
Password	A password.
HostName	The name of the computer on which the INFORMIX-OnLine Dynamic Server or the INFORMIX-SE database server resides.
Service	The name assigned to the Informix database server process running on your UNIX computer. Commonly, the service is <b>sqlexec</b> . Confirm the service name with your system administrator.
ServerName	The name of the database server on which the database that you want to access resides.

### Options That Define Cursor Behavior

The following table lists the types of cursor behavior you can set as default connection options for a data source. The following table lists the initial default that applies if you do not specify a value in the Data-Source Specification section of your **.odbc.ini** file.

Keyword	Attribute
CursorBehavior	<p>A value that determines the type of behavior of the cursors after the transaction ends:</p> <ul style="list-style-type: none"> <li>■ 0 = Closed. This is the default setting.</li> <li>■ 1 = Preserve. Choose this setting to hold the cursors at the current position when the transaction ends. This setting might affect the performance of your database operations.</li> </ul>
EnableScrollable-Cursors	<p>A value that determines if the driver provides scrollable cursors.</p> <ul style="list-style-type: none"> <li>■ 0 = No scrollable cursors. This is the default setting.</li> <li>■ 1 = Enable scrollable cursors. If this value is set, SELECT lists must not include long columns, such as SQL_LONGVARCHAR or SQL_LONGVARBINARY.</li> </ul>



**Tip:** You can also add data sources when you install INFORMIX-CLI. In either case, the steps that you follow are the same.

#### To add a data source

1. Edit your **.odbc.ini** file using a text editor such as **vi**.

If you do not have this file in your home directory, copy the default **odbc.ini** file from the **\$INFORMIXDIR/odbc** directory and change the name to **.odbc.ini**.

2. Under the ODBC Data Sources section, add an entry for your data source.

Each entry in this section lists the data source and a description of the driver that the data source uses. Use the following format for data-source entries:

```
[ODBC Data Sources]
data_source_name=driver_description
.
.
```



<i>data_source_name</i>	identifies the data source that INFORMIX-CLI accesses. You choose the data-source name.
<i>driver_description</i>	describes the driver that the data source accesses. This field is optional. Set this field to Informix Driver.

For example, to associate the **stores7** data source with an INFORMIX-CLI driver, you might make the following entry in the Data Sources section of your **.odbc.ini** file:

```
[ODBC Data Sources]
Stores7=Informix Driver
```

3. After the ODBC Data Sources section, add an entry for each specified data source.

Each data source listed in the ODBC Data Sources section of your **.odbc.ini** file requires a Data Source Specification section. Use the following format for Data Source Specification entries:

```
[data_source_name]
Driver=driver_path
Database=database_name
keyword=attribute
.
.
.
```

<i>data_source_name</i>	is the name of the data source, as specified in the ODBC Data Sources section of your <b>.odbc.ini</b> file.
<i>driver_path</i>	is the full path to your driver.
<i>database_name</i>	is the name of the OnLine or SE database that the data source accesses.  The name can include the Informix database server qualifier.
<i>attribute</i>	specifies the value for the keyword.

For a list of the keywords that the INFORMIX-CLI driver for UNIX supports, see [“Optional Data-Source Configuration Information”](#) on page 2-6.



The required fields in this section are `data_source_name` and `driver_path`.

“Database” is a keyword that is used to define optional connection information. The `database_name` that you enter becomes the default data-source connection. When the database attribute is available on a database server different from the database server identified by the `INFORMIXSERVER` environment variable, the database attribute can be specified as follows:

```
Database=database_name@server_name
```

**Important:** The section name for the Data Source Specification must match the data-source name listed in the ODBC Data Sources section of your `.odbc.ini` file.

For example, the entry for INFORMIX-OnLine Dynamic Server, Version 7.x, for the `stores7` database might look like the following example:

```
[Stores7]
Driver=/usr/informix/cli/dlls/qeinf708.so
Database=stores7
LogonID=Mary
```

In this example, the data source and database are both called `stores7`, and the user ID is `Mary`. You might want the database and data-source names to be the same so that when you connect to a data source, you know the specific database to which you are connecting.



**Tip:** Data-source names are case insensitive. That is, `stores7` and `Stores7` refer to the same data source.

## Sample Data-Source Entry for `.odbc.ini`

A complete `.odbc.ini` data-source entry for the `stores7` data source that is described in the procedure for adding a data source on [page 2-8](#) might look like the following example:

```
[ODBC Data Sources]
Stores7=Informix Driver

[Stores7]
Driver=/usr/informix/cli/dlls/qeinf708.so
Database=stores7
LogonID=Mary
```

## Modifying a Data Source

To edit a data source, use a text editor such as the vi editor. Open your `.odbc.ini` file and modify the appropriate lines in that file. The sections that make up this file are described in [“Adding a Data Source” on page 2-6](#).

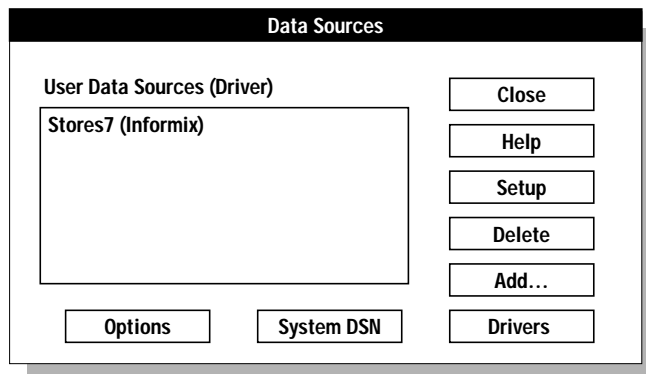
---

## Connecting to a Data Source

An INFORMIX-CLI application can pass connection information in several ways. For example, the application might have the driver prompt the user for connection information, or the application might expect a connection string that specifies the data-source connection. How you connect to a data source depends on the connection method that your INFORMIX-CLI application uses.

## Using Dialog Boxes to Connect to a Data Source

One common way of connecting to a data source is through the Data Source dialog box, as Figure 2-1 illustrates. If your INFORMIX-CLI application is set up to use a dialog box, the Data Sources dialog box appears and prompts you for the appropriate data-source connection information.

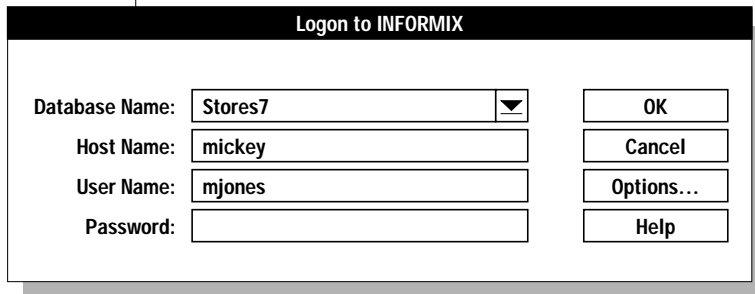


**Figure 2-1**  
The Data Sources  
Dialog Box

To connect to a data source using a dialog box

1. In the **Data Sources** dialog box, select a data source.
2. Click **OK**.

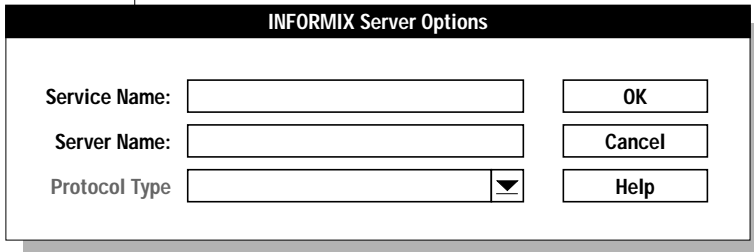
The Informix ODBC Connect dialog box appears, as Figure 2-6 illustrates. The connection information that appears is the default information listed in your `.odbc.ini` file.



**Figure 2-2**  
*The Logon to  
INFORMIX Dialog  
Box*

3. To accept the default values and connect to the data source, click **OK**.  
*or*  
To modify the default values for this data source, continue to step 4.
4. In the **Database Name** text box, type the name of the OnLine or SE database that contains the tables that you want to access.  
You can also click the down arrow to choose a name from a drop-down list.
5. In the **Host Name** text box, type the name of the computer on which your Informix database server resides.
6. In the **User Name** text box, type your user name as specified on the database server.
7. In the **Password** text box, type your password for the UNIX server to which you want to connect.

- To display the INFORMIX Server Options dialog box, click **Options**. This dialog box lets you specify options that are described in “Optional Data-Source Configuration Information” on page 2-6.



The screenshot shows a dialog box titled "INFORMIX Server Options". It features three input fields on the left: "Service Name:", "Server Name:", and "Protocol Type". The "Protocol Type" field is a dropdown menu. To the right of these fields are three buttons: "OK", "Cancel", and "Help".

**Figure 2-3**  
The INFORMIX  
Server Options  
Dialog Box

- Click **OK** to accept any changes you make and return to the Informix ODBC Connect dialog box.
- Click **OK** to complete the logon and to update these values in the **.odbc.ini** file.
- Once the connection information is verified, your application can use the INFORMIX-CLI driver to access the information that the data source contains.

## Using a Connection-String to Connect to a Data Source

Some applications require that you connect to a data source by typing a connection string. The connection string includes several *attributes* that specify how a driver connects to a data source for a particular session. An attribute identifies a specific piece of information that the driver needs to know before it can make the appropriate data-source connection.

A connection string has the following format:

```
"DSN=data_source_name[;attribute=value[;attribute=value]...]"
```

You must specify *data\_source\_name*; however, all other attributes are optional. If you do not specify an attribute, it defaults to the one that is specified in the Data Source Specification section (for the data source specified in the connection string) of your **.odbc.ini** file.

The following table lists the long and short names of the attributes that you can include in the INFORMIX driver connection string. With the exception of the data-source name (which must be specified by DSN), you can use either the long or short names in the connection string. For a description of these attributes, see [“Optional Data-Source Configuration Information” on page 2-6](#)

Attribute Long Name	Short Name
DataSourceName	DSN
Database	DB
HostName	HOST
LogonID	UID
Password	PWD
Service	SERV
CursorBehavior	CB
EnableScrollableCursors	ESC

The following example shows a valid connection string for connecting to the **stores7** data source. The string specifically requests a connection to the **stores7** database that resides on a computer named **rainbow**. The user ID is **mary**.

```
"DSN=Stores7;DB=stores7;HOST=rainbow;SERV=sqlexec;  
UID=mary;PWD=secret"
```

In this example, the service and password are also specified.

---

## INFORMIX-CLI for Windows

INFORMIX-CLI for Windows conforms to the *Microsoft ODBC Specification* (Version 2.5). INFORMIX-CLI supports multiple connections to an OnLine or an SE database. For Windows 3.1, the driver is called **QEINF509.DLL**. For Windows NT and Windows 95, the driver is called **IVINF709.DLL**.

The following sections describe how to set up and configure INFORMIX-CLI so that you can connect your Windows, Windows 95, or Windows NT client application to a data source.

---

## Setting Up INFORMIX-CLI

Perform the following setup activities before you use INFORMIX-CLI:

- Confirm that your system has the appropriate software installed.
- Confirm that the **INFORMIXDIR** environment variable setting reflects the appropriate directory path.

## System Requirements

Depending on your Informix database configuration, you must have a compatible Informix database server installed on your system or network. The following list shows examples of compatible database servers:

- INFORMIX-OnLine (Version 5.x)
- INFORMIX-OnLine Dynamic Server (Version 7.x)
- INFORMIX-SE (Version 5x, 7.x)

For information on these products, refer to your Informix documentation set. For a complete listing of compatible database servers, see the release notes for INFORMIX-CLI.

To take advantage of the GLS feature, you must connect to an Informix database server of Version 7.2 or later. For more information, see the [Guide to GLS Functionality](#). ♦

## Setting the INFORMIXDIR Environment Variable

Before you can use INFORMIX-CLI, make sure that the **INFORMIXDIR** environment variable is set to the full path of the directory where your Informix product is installed.

```
SET INFORMIXDIR=C:\INFORMIX
```

For Windows 3.1 and Windows 95, set this information in the **AUTOEXEC.BAT** file. For Windows NT, set this information in the Registry.

---

## Adding and Modifying Data Sources

A *data source* is a database or file that INFORMIX-CLI accesses. To add and configure data sources, use the ODBC Administrator. The ODBC Administrator then updates your **ODBC.INI** file in Windows 3.1 or your Registry in Windows 95 and Windows NT to reflect your data-source connection information.

### Adding a Data Source

Windows 3.1

In Windows 3.1, the **ODBC.INI** file is an initialization file used by the INFORMIX-CLI and is located in the **WINDOWS** directory. This file contains information about each data source. Before you can connect to a data source, you must use the ODBC Administrator to add the connection information for the data source to the **ODBC.INI** file. For complete information on the format and contents of this file, refer to [“Understanding the odbc.ini File” on page 1-7](#).



**Warning:** *Windows 3.1 users should never modify the **ODBC.INI** file directly. The contents of this file are changed based on the data-source setup and modification you make using the ODBC Administrator. Modifying the **ODBC.INI** file directly might result in data-source configuration errors. ♦*



Windows NT

Windows 95

In Windows NT, **ODBC.INI** is a subkey of the **HKEY\_CURRENT\_USER** key within the Registry. When you access the Registry using this subkey, the ODBC structure is the same as described in the [“File Format for odbc.ini” on page 1-8](#). Before you can connect to a data source, you must use the ODBC Administrator to add the connection information for the data source to the Registry. ♦

### ***Required Data-Source Configuration Information***

When you add a data source, you must provide the name of the data source and the name of the database to which you want to connect to by default. All other connection information is optional.

### ***Optional Data-Source Configuration Information***

When you add a data source, you can choose to define new connection information defaults. You can specify three types of options in your data-source setup:

- Options that define name or location
- Options that define cursor behavior
- Options that define operation handling

#### *Options That Define Name or Location*

The following table lists the types of names and locations you can set as default connection options for a data source. If you do not set a new default, the default value that is listed in the table will apply.

<b>Attribute</b>	<b>Description</b>
Description	A long description that identifies the data source
Default User Name	The host user ID
Host Name	The name of the computer on which the INFORMIX-OnLine Dynamic Server, INFORMIX-OnLine, or INFORMIX-SE database server resides

(1 of 2)

---

Attribute	Description
Service Name	The name assigned to the Informix database server process running on your UNIX computer. Commonly, the service is <b>sqlexec</b> . Confirm the service name with your system administrator.
Server Name	The name of the OnLine or SE database server on which the database that you want to access resides (available only for Windows 95 and Windows NT)
Get DB List from Informix	A value of 0 or 1 that determines from where the driver requests the database list to be returned: <ul style="list-style-type: none"><li data-bbox="541 570 1206 621">■ 1= Request the database list from the database server. This is the default setting.</li><li data-bbox="541 638 1206 690">■ 0 = Use the database list that is specified by the user at driver setup.</li></ul>
Database List	The list of databases that will be displayed in the logon dialog box (that is, the databases to which you will be able to connect) if Get DB List From Informix is set to 0. If more than one name is specified, the names must be separated by commas.

---

(2 of 2)

*Options That Define Cursor Behavior*

The following table lists the types of cursor behavior you can set as default connection options for a data source. If you do not set a new default, the default setting that is listed in the table will apply.

Attribute	Description
Cursor Behavior	<p>A value that determines the type of behavior of the cursors after the transaction ends:</p> <ul style="list-style-type: none"> <li>■ 0 = Closed. This is the default setting.</li> <li>■ 1 = Preserve. Choose this setting to hold the cursors at the current position when the transaction ends. This setting might impact the performance of your database operations.</li> </ul>
Enable Scrollable Cursors	<p>A value that determines if the driver provides scrollable cursors.</p> <ul style="list-style-type: none"> <li>■ 0 = No scrollable cursors. This is the default setting.</li> <li>■ 1 = Enable scrollable cursors. If this value is set, SELECT lists must not include long columns, such as SQL_LONGVARCHAR or SQL_LONGVARBINARY.</li> </ul>

*Options That Define Operation Behavior*

The following table lists the types of operation behavior you can set as default connection options for a data source. If you do not set a new default, the default setting that is listed in the table will apply.

Attribute	Description
Protocol Type	The protocol used to communicate with the database server. Values can be IPX, TCP/IP, or PIPE for Windows 3.1. For Windows NT and Windows 95, values can be <i>xxSOCTCP</i> , <i>xxSOCSPX</i> , or <i>SEIPCPIP</i> where <i>xx</i> represents an abbreviation for the type of database server to which you are connecting. For example, <i>SESOCSPX</i> indicates that you are using an INFORMIX-SE with sockets and IPX.
Yield Proc	<p>A value of 0, 1, 2, or 3 that determines whether you can work in other Windows applications when the INFORMIX-CLI driver is busy.</p> <ul style="list-style-type: none"> <li>■ 0 = Peek and dispatch. This setting causes the Informix driver to check the Windows message queue and send any messages to the appropriate Windows application.</li> <li>■ 1 = No yielding. This setting does not let you work in other applications. This is the default setting.</li> <li>■ 2 = Informix yield procedure. This is the default setting. For more information, refer to the section on the <code>TMBLOCKMODE</code> environment variable in the <i>INFORMIX-DCE/NET User's Guide</i>.</li> <li>■ 3 = Dispatch via Windows Yield function. This setting turns control over to the Windows kernel. The Windows kernel checks the message queue and sends any messages to the appropriate application window.</li> </ul> <p>The recommended value is 1. ♦</p>

**Windows 3.1**

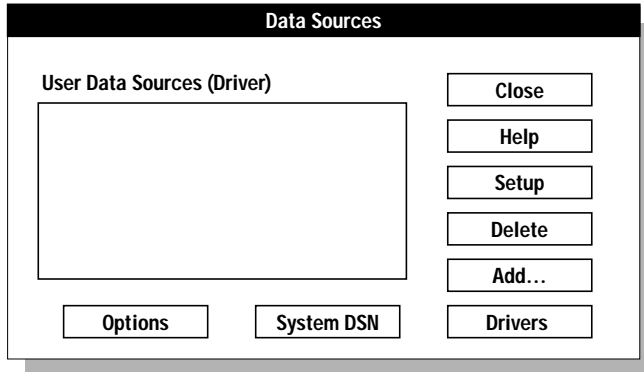


**Tip:** You can also add data sources when you install *INFORMIX-CLI*. In either case, the steps that you follow are the same.

**To add a data source**

1. Invoke the ODBC Administrator.

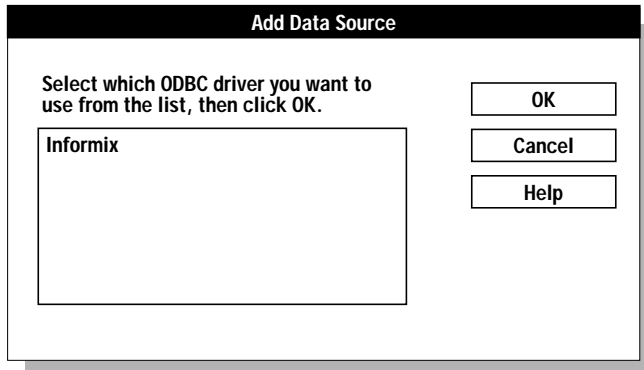
The Data Sources dialog box appears, as Figure 2-4 illustrates.



**Figure 2-4**  
*The Data Sources Dialog Box*

2. Click **Add**.

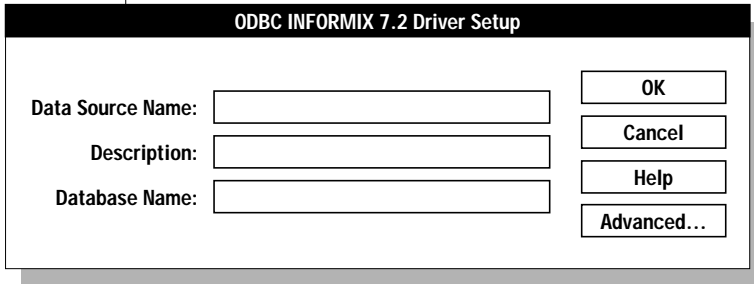
The Add Data Source dialog box appears, Figure 2-5 illustrates.



**Figure 2-5**  
*The Add Data Source Dialog Box*

3. Select the Informix driver from the Installed ODBC Drivers list.
4. Click **Add**.

The ODBC INFORMIX Driver Setup dialog box appears, as Figure 2-6 illustrates.



**Figure 2-6**  
*The ODBC  
INFORMIX Driver  
Setup Dialog Box*

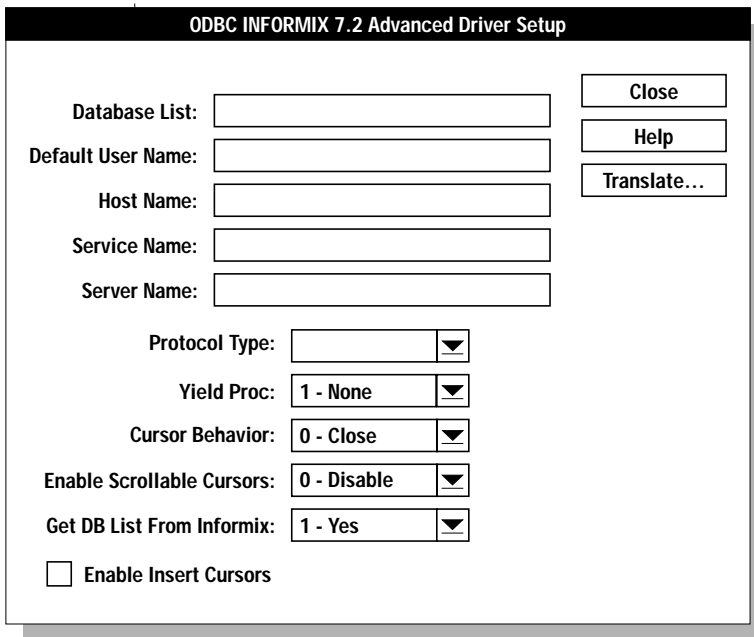
In the **Data Source Name** text box, enter the name of the data source you want to access.

You define the data-source name; that is, it can be any name that you choose.

5. In the **Description** text box, type a long description of your data source. This section is optional.
6. In the **Database Name** text box, enter the name of the database to which you want to connect by default.

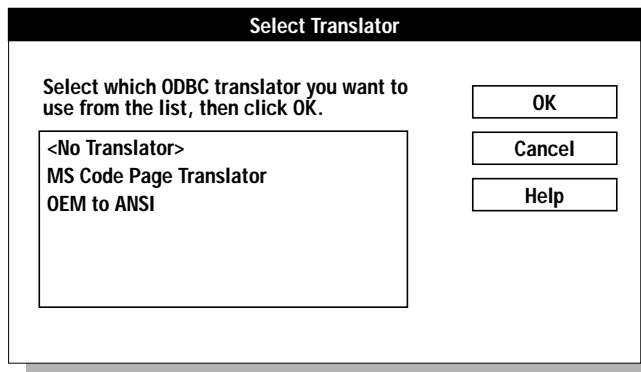
You now have entered enough information to be able to connect to the data source:

- ❑ To enter the data source, click **OK**.  
The Data Sources dialog box appears, as Figure 2-4 illustrates. You can then click **Add** to add another data source or click **Close** to exit the Data Sources dialog box.
- ❑ To add optional connection information about the data source, click **Advanced**.  
The ODBC INFORMIX 7.2 Advanced Driver Setup Window appears, as Figure 2-7 illustrates.



**Figure 2-7**  
The ODBC  
INFORMIX  
Advanced Driver  
Setup Dialog Box

7. Type the information you want to specify about this data source. You can also choose options from the drop-down lists. For a description of the advanced connection options that are available, see [“Optional Data-Source Configuration Information”](#) on page 2-17.
8. To specify a translator, click **Translate**.  
The Translator dialog box appears, as Figure 2-8 illustrates.



**Figure 2-8**  
The Translator  
Dialog Box

9. Select the translator you want from the list.
10. Click **OK** to return to the ODBC INFORMIX Advanced Driver Setup dialog box.
11. To accept your specifications for this data source, click **OK**.  
When you click **OK** in the ODBC INFORMIX Advanced Driver Setup dialog box, the Data Sources dialog box appears, as shown in Figure 2-4.
12. To add another data source, click **Add**. To exit the Data Sources dialog box, click **Close**.

After you click **OK** in the ODBC INFORMIX Driver Setup dialog box, the ODBC Administrator updates your **ODBC.INI** file (or Registry). The information that you entered in any of the setup dialog boxes becomes the new default data-source connection information for this data source.

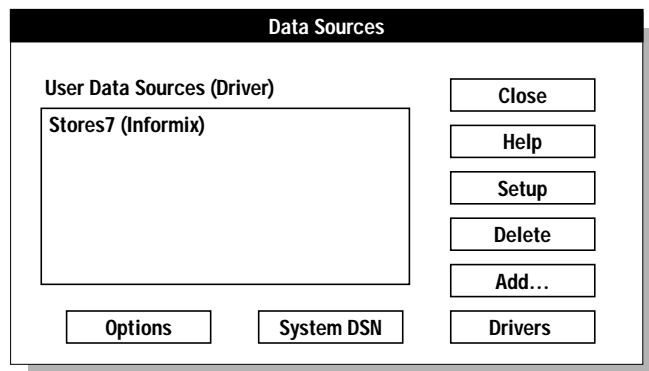
## Modifying a Data Source

Use the ODBC Administrator to make all modifications to your data source. To make basic modifications to the defaults for a data source, see [“Using Dialog Boxes to Connect to a Data Source” on page 25](#). To make more detailed modifications to a data source, complete the following steps.

### To modify a data source

1. Start the ODBC Administrator.

The Data Sources dialog box appears, as Figure 2-9 illustrates.

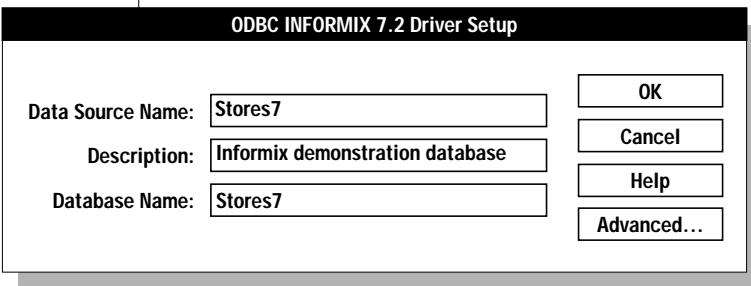


**Figure 2-9**  
*The Data Sources  
Dialog Box*



2. In the Data Sources dialog box, select the Informix data source that you want to modify and then click **Setup**.

The ODBC INFORMIX Driver Setup dialog box appears, as Figure 2-10 illustrates. The values that appears the default entries specified for this data-source connection.



**Figure 2-10**  
A Completed ODBC  
INFORMIX Driver  
Setup Dialog Box

3. Modify the applicable data-source text boxes. For more information regarding available options, see [“Adding a Data Source” on page 2-16](#).
4. When you are finished, click **OK** in the ODBC INFORMIX Driver Setup dialog box.

The ODBC Administrator updates your **ODBC.INI** file.

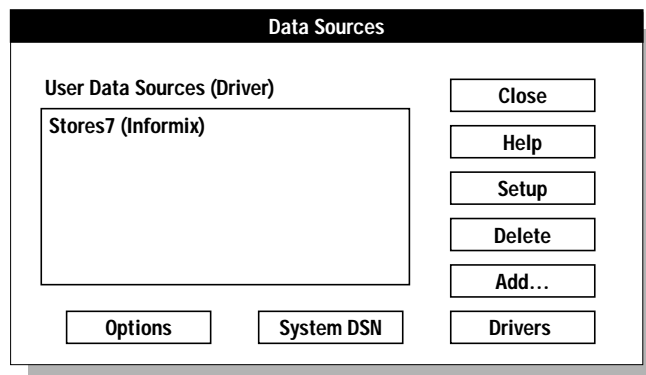
When you connect to this data source using either a dialog box or connection string, the values that you entered appear as the new default entries for the data-source connection.

## Connecting to a Data Source

An INFORMIX-CLI application can pass connection information in several ways. For example, the application might have the driver always prompt the user for connection information, or the application might expect a connection string that specifies the data-source connection. How you connect to a data source depends on the connection method that your INFORMIX-CLI application uses.

## Using Dialog Boxes to Connect to a Data Source

One common way of connecting to a data source is through the Data Sources dialog box, as Figure 2-11 illustrates. If your INFORMIX-CLI application is set up to use a dialog box, the Data Sources dialog box appears and prompts you for the appropriate data-source connection information.

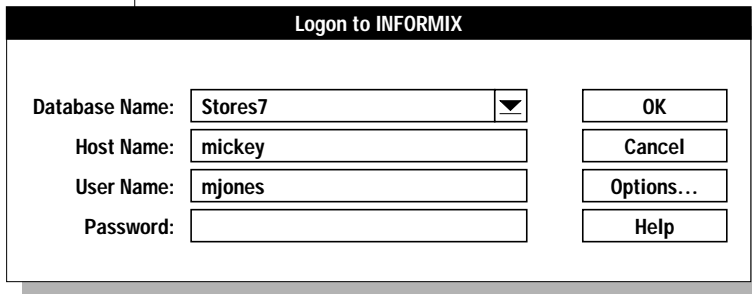


**Figure 2-11**  
*The Data Sources  
Dialog Box*

**To connect to a data source using a dialog box**

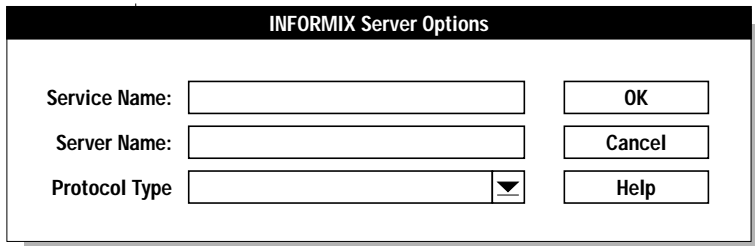
1. In the **Data Sources** dialog box, select a data source.
2. Click **OK**.

The Logon to Informix dialog box appears, as Figure 2-12 illustrates. The connection information that appears is the default information listed in your **ODBC.INI**.



**Figure 2-12**  
*The Logon to  
INFORMIX Dialog  
Box*

3. To accept the default values and connect to the data source, click **OK**.  
or  
To modify the default values for this data source, continue to step 4.
4. In the **Database Name** text box, type the name of the OnLine or SE database that contains the tables that you want to access.  
You can also click the down arrow to choose a name from a drop-down list.
5. In the **Host Name** text box, type the name of the computer on which your Informix database server resides.
6. In the **User Name** text box, type your user name as specified on the database server.
7. In the **Password** text box, type your password for the UNIX server to which you want to connect.
8. To display the INFORMIX Server Options dialog box, click **Options**.  
This dialog box lets you specify options that are described in [“Optional Data-Source Configuration Information”](#) on page 2-17.



**Figure 2-13**  
The INFORMIX  
Server Options  
Dialog Box

9. Click **OK** to accept any changes you make and return to the Logon to Informix dialog box.
10. Click **OK** to complete the logon and to update these values in **ODBC.INI**.

Once the connection information is verified, your application can use the INFORMIX-CLI driver to access the information that the data source contains.

## Using a Connection-String to Connect to a Data Source

Some applications require that you connect to a data source by typing a connection string. The connection string includes several *attributes* that specify how a driver connects to a data source for a particular session. An attribute identifies a specific piece of information that the driver needs to know before it can make the appropriate data-source connection.

A connection string has the following format:

```
"DSN=data_source_name[;attribute=value[;attribute=value]...]"
```

You must specify *data\_source\_name*; however, all other attributes are optional. If you do not specify an attribute, it defaults to the one that is specified in the Data Source Specification section (for the data source specified in the connection string) of your **ODBC.INI** file in Windows 3.1 or your Registry in Windows NT and Windows 95.

The following table lists the long and short names of the attributes that you can include in the INFORMIX driver connection string. With the exception of the data source name (which must be specified by DSN), you can use either the long or short names in the connection string. For a description of these attributes, see [“Required Data-Source Configuration Information”](#) and [“Optional Data-Source Configuration Information”](#) on page 2-17.

Attribute	Short Name
DataSourceName	DSN
Database	DB
HostName	HOST
LogonID	UID
Password	PWD
Service	SERV
Server	SERVER
GetDBListFrom Informix	GDBLFI
CursorBehavior	CB
EnableScrollableCursors	ESC
Protocol	PRO (Windows 3.1 and Windows NT only)
YieldProc	YLD (Windows 3.1 only)

The following example shows a valid connection string for connecting to the **stores7** data source. The string specifically requests a connection to the **stores7** database that resides on a computer named **rainbow**. The user ID is **mary**.

```
"DSN=Stores7;DB=stores7;HOST=rainbow;SERV=sqlexec;  
UID=mary;PWD=secret"
```

In this example, the service and password are also specified.



---

# Guidelines for Calling INFORMIX-CLI Functions

Determining INFORMIX-CLI Conformance Levels . . . . .	3-3
Using the Driver Manager . . . . .	3-4
Calling Functions . . . . .	3-5
Buffers . . . . .	3-5
Input Buffers . . . . .	3-5
Output Buffers . . . . .	3-6
Environment, Connection, and Statement Handles . . . . .	3-7
Using Data Types . . . . .	3-8
Function Return Codes . . . . .	3-8





**T**his chapter describes the general characteristics of INFORMIX-CLI functions, determining driver conformance levels, the role of the driver manager, passing function arguments to a driver, and the values that functions return.

Each INFORMIX-CLI function name starts with the prefix **SQL**. Each function accepts one or more arguments. Arguments are defined as input (to the driver) or output (from the driver).

C programs that call INFORMIX-CLI functions must include the **sql.h** and **sqlext.h** header files. These files define INFORMIX-CLI constants and types and provide function prototypes for all INFORMIX-CLI functions.

---

## Determining INFORMIX-CLI Conformance Levels

ODBC defines conformance levels for drivers in two areas: the ODBC API and the ODBC SQL grammar, which includes the ODBC SQL data types. These levels establish standard sets of functionality. By verifying the conformance levels supported by a driver, an application can determine if the driver provides the necessary functionality.

To determine the function conformance level of a driver, an application calls **SQLGetInfo** with the **SQL\_ODBC\_SAG\_CLI\_CONFORMANCE** and **SQL\_ODBC\_API\_CONFORMANCE** flags.

To determine the SQL conformance level of a driver, an application calls **SQLGetInfo** with the **SQL\_ODBC\_SQL\_CONFORMANCE** flag. To determine whether a driver supports a specific SQL extension, an application calls **SQLGetInfo** with a flag for that extension. To determine whether a driver supports a specific SQL data type, an application calls **SQLGetTypeInfo**.

---

## Using the Driver Manager

The driver manager is a shared library that provides access to the Informix driver. An INFORMIX-CLI application typically links with the driver manager import library to gain access to the driver manager. For UNIX, this file is named **libodbc** with platform-specific extensions. For Windows 3.1, this file is named **ODBC.DLL** and for Windows 95 and Windows NT, this file is named **ODBC32.DLL**.

Whenever an INFORMIX-CLI application calls a function, the driver manager performs one of the following actions:

- For **SQLDataSources** and **SQLDrivers**, the driver manager processes the call. It does not pass the call to the driver.
- For **SQLGetFunctions**, the driver manager passes the call to the driver associated with the connection.
- For **SQLAllocEnv**, **SQLAllocConnect**, **SQLSetConnectOption**, **SQLFreeConnect**, and **SQLFreeEnv**, the driver manager processes the call. The driver manager calls **SQLAllocEnv**, **SQLAllocConnect**, and **SQLSetConnectOption** in the driver when the application calls a function to connect to the data source (**SQLConnect**, **SQLDriverConnect**, or **SQLBrowseConnect**). The driver manager calls **SQLFreeConnect** and **SQLFreeEnv** in the driver when the application calls **SQLDisconnect**.
- For **SQLConnect**, **SQLDriverConnect**, **SQLBrowseConnect**, and **SQLError**, the driver manager performs initial processing and then passes the call to the driver associated with the connection.

For any other function, the driver manager passes the call to the driver.

If requested, the driver manager records each called function in a trace file. The name of each function is recorded, along with the values of the input arguments and the names of the output arguments (as listed in the function definitions).<sup>1</sup>

---

## Calling Functions

The following paragraphs describe general characteristics of INFORMIX-CLI functions.

### Buffers

An application passes data to the driver in an input buffer. The driver returns data to the application in an output buffer. The application must allocate memory for both input and output buffers. (If the application uses the buffer to retrieve string data, the buffer must contain space for the null termination byte.)

Some functions accept pointers to buffers that are used later by other functions. The application must ensure that these pointers remain valid until all applicable functions have used them. For example, the argument *rgbValue* in **SQLBindCol** points to an output buffer where **SQLFetch** returns the data for a column.

### *Input Buffers*

An application passes the address and length of an input buffer to the driver. The length of the buffer must be one of the following values:

- A length greater than or equal to zero  
This value is the actual length of the data in the input buffer. For character data, a length of zero indicates that the data is an empty (zero length) string. A length of zero is different from a null pointer. If the application specifies the length of character data, the character data does not need to be null-terminated.
- **SQL\_NTS**  
This value specifies that a character data value is null terminated.
- **SQL\_NULL\_DATA**  
This value tells the driver to ignore the value in the input buffer and use a NULL data value instead. It is valid only when the input buffer provides the value of a parameter in an SQL statement.

The operation of INFORMIX-CLI functions on character data that contains embedded null characters is undefined and is not recommended for maximum interoperability.

Unless it is specifically prohibited in a function description, the address of an input buffer can be a null pointer. In this case, the value of the corresponding buffer-length argument is ignored.

For more information about input buffers, see [“Converting Data from C to SQL Data Types” on page C-24](#).

### ***Output Buffers***

An application passes the following arguments to the driver, so that the driver can return data in an output buffer:

- The address of the buffer in which the driver returns the data (the output buffer)

Unless it is specifically prohibited in a function description, the address of an output buffer can be a null pointer. In this case, the driver does not return anything in the buffer and, in the absence of other errors, returns SQL\_SUCCESS.

If necessary, the driver converts data before returning it. The driver always null-terminates character data before returning it.

- The length of the buffer

This value is ignored by the driver if the returned data has a fixed length in C, such as an integer, real number, or date structure.

- The address of a variable in which the driver returns the length of the data (the length buffer)

The returned length of the data is SQL\_NULL\_DATA if the data is a NULL value in a result set. Otherwise, it is the number of bytes of data that are available to return. If the driver converts the data, it is the number of bytes that remain after the conversion. For character data, it does not include the null-termination byte added by the driver.

If the output buffer is too small, the driver attempts to truncate the data. If the truncation does not cause a loss of significant data, the driver returns the truncated data in the output buffer, returns the length of the available data (as opposed to the length of the truncated data) in the length buffer, and returns `SQL_SUCCESS_WITH_INFO`. If the truncation causes a loss of significant data, the driver leaves the output and length buffers untouched and returns `SQL_ERROR`. The application calls `SQLError` to retrieve information about the truncation or the error.

For more information about output buffers, see [“Converting Data from SQL to C Data Types” on page C-13](#).

## Environment, Connection, and Statement Handles

The driver manager and the driver allocate storage for information about the environment, each connection, and each SQL statement when an application requests. The handles to these storage areas are returned to the application, which uses one or more handles in each call to a function.

The INFORMIX-CLI interface uses the following types of handles that are defined by the ODBC specification:

- The *environment handle* identifies memory storage for global information, including the valid connection handles and the current active connection handle. The environment handle is an `HENV` variable type. An application uses a single environment handle; it must request this handle before it connects to a data source.
- *Connection handles* identify memory storage for information about particular connections. A connection handle is an `HDBC` variable type. An application must request a connection handle before it connects to a data source. Each connection handle is associated with the environment handle. However, the environment handle can have multiple connection handles associated with it.
- *Statement handles* identify memory storage for information about SQL statements. A statement handle is an `HSTMT` variable type. An application must request a statement handle before it submits SQL requests. Each statement handle is associated with exactly one connection handle. However, each connection handle can have multiple statement handles associated with it.

For more information about requesting a connection handle, see [“Connecting to a Data Source” on page 5-5](#). For more information about requesting a statement handle, see [“Executing an SQL Statement” on page 6-5](#).

## Using Data Types

Data stored on a data source has an SQL data type. The INFORMIX-CLI driver maps Informix-specific SQL data types to ODBC SQL data types, which are defined in the ODBC SQL grammar. (The driver returns these mappings through **SQLGetTypeInfo**. It also uses the ODBC SQL data types to describe the data types of columns and parameters in **SQLColAttributes** and **SQLDescribeCol**.)

Each SQL data type corresponds to an ODBC C data type. By default, the driver assumes that the C data type of a storage location corresponds to the SQL data type of the column or parameter to which the location is bound. If the C data type of a storage location is not the *default* C data type, the application can specify the correct C data type with the *fCType* argument in **SQLBindCol**, **SQLGetData**, or **SQLBindParameter**. Before the driver returns data from the data source, it converts the data to the specified C data type. Before the driver sends data to the data source, it converts the data from the specified C data type.

For more information about data types, see [Appendix C](#). The C data types are defined in the **sql.h** and **sqlext.h** header files.

## Function Return Codes

When an INFORMIX-CLI application calls a function, the driver executes the function and returns a predefined code. The following return codes indicate success, warning, or failure status:

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

When the function returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, the application can call **SQLERROR** to retrieve additional information about the error. For a complete description of return codes and error handling, see [Chapter 8, “Retrieving Status and Error Information.”](#)





---

# Basic Application Steps

How an Application Uses the INFORMIX-CLI Interface . . . . . 4-3



**T**his chapter describes how an application uses the INFORMIX-CLI interface to interact with a data source. Chapters 5 through 9 discuss the individual actions that an application performs as it interacts with a data source and [Chapter 10, “Constructing an INFORMIX-CLI Application,”](#) provides code examples of such basic applications. [Chapter 11, “Designing Performance-Oriented Applications,”](#) offers design and coding suggestions that can enhance application performance.

---

## How an Application Uses the INFORMIX-CLI Interface

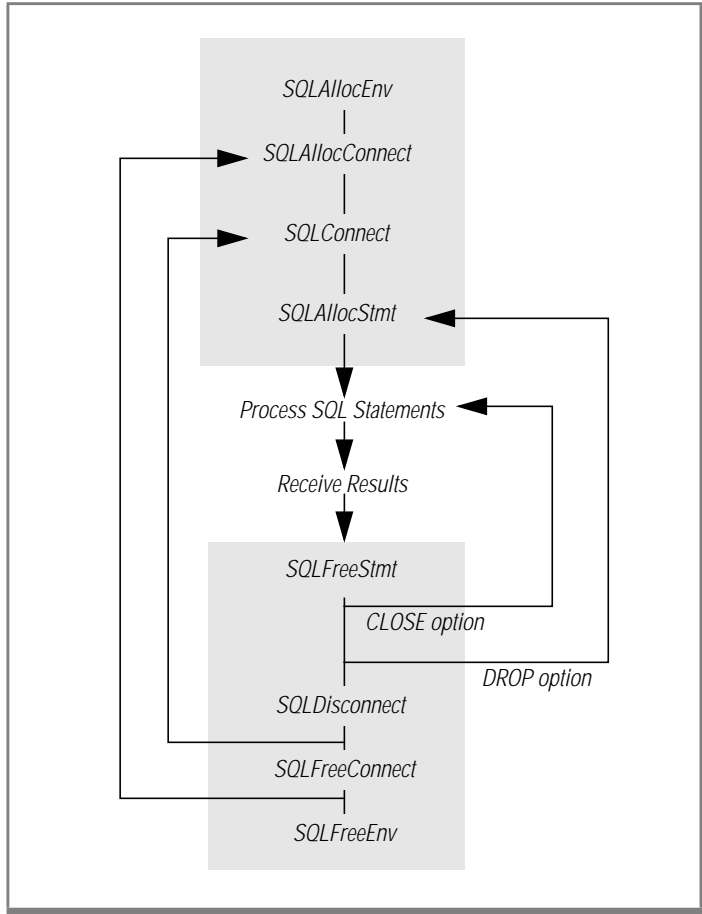
An application uses the INFORMIX-CLI interface to make a connection to a data source, issue SQL statements to a data source, process result data dynamically, and terminate a connection.

### To interact with a data source

1. Connect to the data source, specifying the data-source name and any additional information needed to complete the connection.

2. Process one or more SQL statements:
  - The application places the SQL text string in a buffer. If the statement includes parameter markers, it sets the parameter values.
  - If the statement returns a result set, the application assigns a cursor name for the statement or allows the driver to assign one.
  - The application submits the statement for prepared or immediate execution.
  - If the statement creates a result set, the application can inquire about the attributes of the result set, such as the number of columns and the name and type of a specific column. It assigns storage for each column in the result set and fetches the results.
  - If the statement causes an error, the application retrieves error information from the driver and takes appropriate action.
3. End each transaction by committing it or rolling it back.
4. Terminate the connection when it finishes interacting with the data source.

Figure 4-1 shows the function calls that a basic application makes to connect to a data source, process SQL statements, and disconnect from the data source. Depending on your needs, your application can call other functions.



**Figure 4-1**  
A Sample Listing of  
Function Calls Made  
by an INFORMIX-  
CLI Application



---

# Connecting to a Data Source

Description of Data Sources . . . . .	5-3
Initializing the Environment . . . . .	5-4
Allocating a Connection Handle . . . . .	5-5
Connecting to a Data Source . . . . .	5-5
Connecting with <code>SQLConnect</code> . . . . .	5-5
Connecting to a Data Source with <code>SQLDriverConnect</code> . . . . .	5-6
Connection Browsing with <code>SQLBrowseConnect</code> . . . . .	5-9
Translating Data . . . . .	5-9
Related Functions . . . . .	5-10





**T**his chapter describes how to establish a connection to a data source.

---

## Description of Data Sources

A data source consists of the data that a user wants to access, its associated DBMS, the platform on which the DBMS resides, and the network (if any) used to access that platform. Each data source requires that the driver provide connection information. For a description of the connection information that the INFORMIX-CLI driver provides, see [“Adding a Data Source” on page 2-6](#) for UNIX platforms or [“Adding a Data Source” on page 2-16](#) for Windows environments.

The connection information for each data source is stored in the **odbc.ini** file, which is created during installation and maintained with a text editor or an administration program. A section in this file lists the available data sources. Additional sections describe each data source in detail, specifying the driver name, a description, and any additional information the driver needs to connect to the data source.

For example, suppose a user has two data sources: **Personnel**, which uses an INFORMIX-SE database server, and **Payroll**, which uses an INFORMIX-OnLine Dynamic Server database server. The section that lists the data sources might appear as follows:

```
[ODBC Data Sources]
Personnel=Informix SE
Payroll=Informix OnLine
```

The section that describes the **Personnel** data source might appear as follows:

```
[Personnel]
Driver=/usr/informix/cli/dlls/qeinf708.so
Description=Personnel database
Database=personnel
```

For more information about the `odbc.ini` file, see [“Understanding the odbc.ini File” on page 1-7](#).

---

## Initializing the Environment

Before an application can use any other function, it must initialize the INFORMIX-CLI interface and associate an environment handle with the environment. To initialize the interface and allocate an environment handle, an application performs the following operations:

1. Declares a variable of type `HENV`.

For example, your application could use the following declaration:

```
HENV henv1;
```

2. Calls `SQLAllocEnv` and passes it the address of the variable.

The driver initializes the environment, allocates memory to store information about the environment, and returns the environment handle in the variable.

These steps should be performed only once by your application; `SQLAllocEnv` supports one or more connections to data sources.

---

## Allocating a Connection Handle

Before your INFORMIX-CLI application can connect to the driver, you must allocate a connection handle for the connection. To allocate a connection handle, an application performs the following operations:

1. Declares a variable of type `HDBC`.  
For example, the application could use the following declaration:  

```
HDBC hdbc1;
```
2. Calls `SQLAllocConnect` and passes it the address of the variable.  
The driver allocates memory to store information about the connection and returns the connection handle in the variable.

---

## Connecting to a Data Source

After allocating a connection handle, you must specify a specific driver (the Informix driver) and data source. INFORMIX-CLI provides functions that allow different methods for connecting to a data source.

### Connecting with `SQLConnect`

Your INFORMIX-CLI application passes the following information to the driver in a call to `SQLConnect`:

- Data-source name is the name of the data source being requested by the application.
- UserID is the login ID or account name for access to the data source, if appropriate (optional).
- Passwords is a character string associated with the user ID that allows access to the data source (optional).

When your application calls `SQLConnect`, the driver manager uses the data-source name to read the name of the driver shared library from the appropriate section of the `odbc.ini` file. It then loads the driver shared library and passes the `SQLConnect` arguments to it. If the driver needs additional information to connect to the data-source, it reads this information from the same section of the `odbc.ini` file.

If the application specifies a data-source name that is not in the `odbc.ini` file, or if the application does not specify a data-source name, the driver manager searches for the default data-source specification. If it finds the default data-source, it loads the default driver shared library and passes the application-specified data source name to it. If no default data source exists, the driver manager returns an error.

## Connecting to a Data Source with `SQLDriverConnect`

Your application can connect to a data source by calling `SQLDriverConnect`.

`SQLDriverConnect` supports the following:

- Data sources that require more connection information than the three arguments in `SQLConnect`
- Dialog boxes to prompt the user for all connection information
- Data sources that are not defined in the `odbc.ini` file

`SQLDriverConnect` uses a connection string to specify the information needed to connect to a driver and data source.

The connection string contains the following information:

- Data-source name or driver description
- Zero or more user IDs
- Zero or more passwords
- Zero or more data-source-specific parameter values

The connection string is a more flexible interface than the data-source name, user ID, and password used by `SQLConnect`. The application can use the connection string for multiple levels of login authorization or to convey other data source-specific connection information.

An application calls `SQLDriverConnect` in one of the following ways:

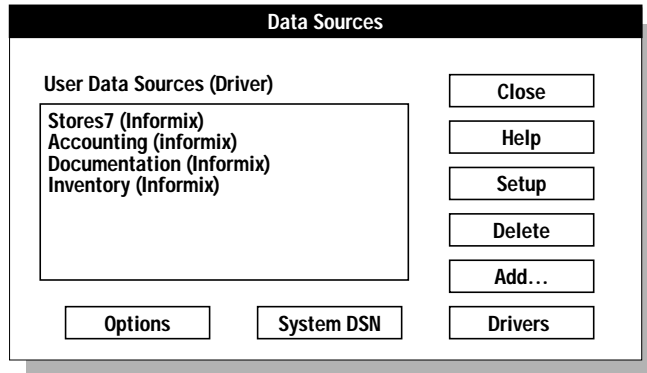
- Specifies a connection string that contains a data-source name  
The driver manager retrieves the full path of the driver shared library associated with the data-source from the `odbc.ini` file. To retrieve a list of data-source names, an application calls `SQLDataSources`.
- Specifies a connection string that contains a driver description  
The driver manager retrieves the full path of the driver shared library. To retrieve a list of driver descriptions, an application calls `SQLDrivers`.
- Specifies a connection string that does not contain a data-source name or a driver description  
The driver manager displays a dialog box from which the user selects a data-source name. The driver manager then retrieves the full path of the driver shared library associated with the data-source.

The driver manager then loads the driver shared library and passes the `SQLDriverConnect` arguments to it.

After the driver connects to the data source, it returns the connection information to the application. The application might store this information for future use.

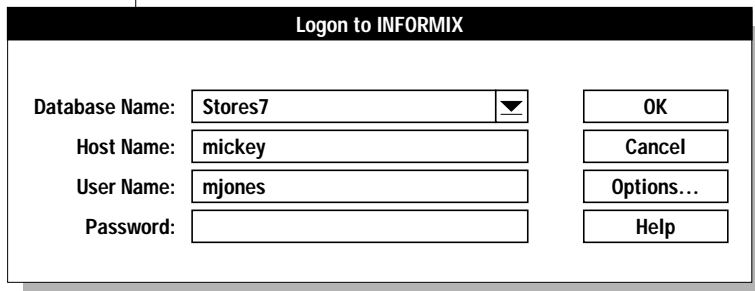
If the application specifies a data-source name that is not in the `odbc.ini`, the driver manager searches for the default data-source specification. If it finds the default data source, it loads the default driver shared library and passes the application-specified data-source name to it. If no default data source exists, the driver manager returns an error.

When the application calls **SQLDriverConnect** and requests that the user be prompted for information, the driver manager displays the SQL Data Sources dialog box, as Figure 5-1 illustrates.



**Figure 5-1**  
*The SQL Data Sources Dialog Box*

When an application requests it, the driver displays the Logon to INFORMIX dialog box to retrieve login information, as Figure 5-2 illustrates.



**Figure 5-2**  
*The Logon to INFORMIX Dialog Box*

## Connection Browsing with SQLBrowseConnect

**SQLBrowseConnect** supports an iterative method of listing and specifying the attributes and attribute values required to connect to a data source. For each level of a connection, an application calls **SQLBrowseConnect** and specifies the connection attributes and attribute values for that level. First level connection attributes always include the data-source name or driver description; the connection attributes for later levels are data-source dependent but might include the host, user name, and database.

Each time an application calls **SQLBrowseConnect**, it validates the current attributes, returns the next level of attributes, and returns a user-friendly name for each attribute. It might also return a list of valid values for those attributes: After an application has specified each level of attributes and values, **SQLBrowseConnect** connects to the data source and returns a complete connection string. This string can be used with **SQLDriverConnect** to connect to the data source at a later time.

## Translating Data

An INFORMIX-CLI application and a data source can store data in different formats. For example, the application might use a different character set than the one the data source uses. The driver can translate all data (data values, SQL statements, table names, row counts, and so on) that passes between the driver and the data source.

The driver translates data by calling functions in a translation shared library. A default translation shared library can be specified for the data source in the **odbc.ini** file; the application can override this by calling **SQLSetConnectOption**. When the driver connects to the data source, it loads the translation shared library if one has been specified. After the driver has connected to the data source, the application might specify a new translation shared library by calling **SQLSetConnectOption**. For more information about specifying a default translation shared library, see [“ODBC Options” on page 1-10](#).

Translation functions might support several types of translation. For example, a function that translates data from one character set to another might support a variety of character sets. To specify a particular type of translation, an application can pass an option flag to the translation functions with **SQLSetConnectOption**.

---

## Related Functions

The following functions are related to connections, drivers, and data sources. For more information about these functions, see [Chapter 13, “INFORMIX-CLI Function Reference.”](#)

---

Function	Description
<b>SQLDataSources</b>	Retrieves a list of available data sources. The driver manager retrieves this information from the <b>odbc.ini</b> file. An application can present this information to a user or automatically select a data source.
<b>SQLDrivers</b>	Retrieves a list of installed drivers and their attributes. The driver manager retrieves this information from the <b>odbcinst.ini</b> file. An application can present this information to a user or automatically select a driver.
<b>SQLGetFunctions</b>	Retrieves functions supported by a driver. This function allows an application to determine at runtime whether a particular function is supported by a driver.
<b>SQLGetInfo</b>	Retrieves general information about a driver and data source, including filenames, versions, conformance levels, and capabilities.
<b>SQLGetTypeInfo</b>	Retrieves the SQL data types supported by a driver and data source.
<b>SQLSetConnectOption</b> <b>SQLGetConnectOption</b>	Sets or retrieves connection options, such as the data-source access mode, automatic transaction commitment, time-out values, function tracing, data-translation options, and transaction isolation.

---



---

# Executing SQL Statements

Allocating a Statement Handle . . . . .	6-5
Executing an SQL Statement . . . . .	6-5
Prepared Execution . . . . .	6-6
Direct Execution . . . . .	6-7
Setting Parameter Values . . . . .	6-7
Performing Transactions . . . . .	6-9
Retrieving Information About the Data-Source Catalog . . . . .	6-9
Sending Parameter Data at Execution Time . . . . .	6-10
Specifying Arrays of Parameter Values . . . . .	6-11
Using ODBC Extensions to SQL . . . . .	6-12
Date, Time, and Time-Stamp Data . . . . .	6-13
Scalar Functions . . . . .	6-14
LIKE Predicate Escape Characters . . . . .	6-15
Outer Joins . . . . .	6-16
Procedures . . . . .	6-17
Related Functions . . . . .	6-19



**A**n application can submit any SQL statement that is supported by a data source. ODBC defines a standard syntax for SQL statements (see [Appendix B](#)). For maximum interoperability, an application should submit only SQL statements that use this syntax; the driver translates these statements to the syntax used by the data source. If an application submits an SQL statement that does not use the ODBC syntax, the driver passes it directly to the data source.

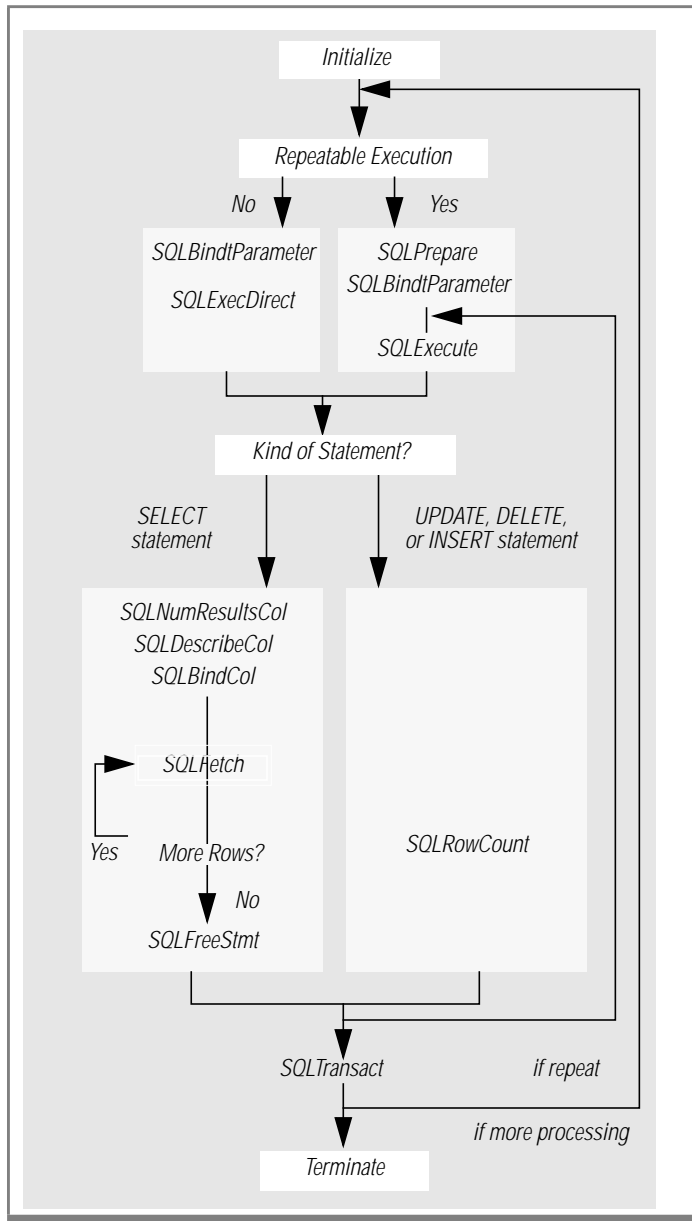


***Important:** For `CREATE TABLE` and `ALTER TABLE` statements, applications should use the data type name returned by `SQLGetTypeInfo` in the `TYPE_NAME` column, rather than the data type name defined in the SQL grammar.*

Statements can be executed a single time with `SQLExecDirect` or prepared with `SQLPrepare` and executed multiple times with `SQLExecute`. An application calls `SQLTransact` to commit or roll back a transaction.

Figure 6-1 illustrates a simple sequence of function calls to execute SQL statements.

**Figure 6-1**  
*A Sequence of Function Calls to Execute SQL Statements*



---

## Allocating a Statement Handle

Before your application can submit an SQL statement, you must allocate a statement handle for the statement. To allocate a statement handle, an application performs the following operations:

1. Declares a variable of type `HSTMT`.

For example, the application could use the following declaration:

```
HSTMT hstmt1;
```

2. Calls `SQLAllocStmt` and passes it the address of the variable and the connected `hdbc` with which to associate the statement.

The driver allocates memory to store information about the statement, associates the statement handle with the `hdbc`, and returns the statement handle in the variable.

---

## Executing an SQL Statement

Your application can submit an SQL statement for execution in the following ways:

- *Prepared* statements call `SQLPrepare` and then call `SQLExecute`.
- *Direct* statements call `SQLExecDirect`.

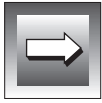
These options are similar, but not identical to, the prepared and immediate options in embedded SQL. For a comparison of the ODBC functions and embedded SQL, see [Appendix D](#).

## Prepared Execution

Prepare a statement before you execute it if either of the following situations is true:

- The application can execute the statement more than once, possibly with intermediate changes to parameter values.
- The application needs information about the result set prior to execution.

A prepared statement executes faster than an unprepared statement because the data source compiles the statement, produces an access plan, and returns an access plan identifier to the driver. The data source minimizes processing time because it does not have to produce an access plan each time it executes the statement. Network traffic is minimized because the driver sends the access plan identifier, instead of the entire statement, to the data source.



**Important:** *Committing or rolling back a transaction, either by calling `SQLTransact` or by using the `SQL_AUTOCOMMIT` connection option, can cause the data source to delete the access plans for all hstmts on an hdbc. For more information, see “`SQL_CURSOR_COMMIT_BEHAVIOR`” and “`SQL_CURSOR_ROLLBACK_BEHAVIOR`” on page 13-192.*

To prepare and execute an SQL statement, an application performs the following operations:

1. Calls **SQLPrepare** to prepare the statement.
2. Sets the values of any statement parameters.  
For more information, see [“Setting Parameter Values” on page 6-7](#).
3. Retrieves information about the result set, if necessary.  
For more information, see [“Determining the Characteristics of a Result Set” on page 7-4](#).
4. Calls **SQLExecute** to execute the statement.
5. Repeats steps 2 through 4 as necessary.

## Direct Execution

Execute a statement directly if both of the following situations are true:

- The application executes the statement only once.
- The application does not need information about the result set prior to execution.

To execute an SQL statement directly, an application performs the following operations:

1. Sets the values of any statement parameters.  
For more information, see [“Setting Parameter Values.”](#)
2. Calls `SQLExecDirect` to execute the statement.

---

## Setting Parameter Values

An SQL statement can contain parameter markers that indicate values that the driver retrieves from the application at execution time. For example, an application might use the following statement to insert a row of data into the `EMPLOYEE` table:

```
INSERT INTO EMPLOYEE (NAME, AGE, HIREDATE) VALUES (?, ?, ?)
```

An application uses parameter markers instead of literal values when one of the following situations occurs:

- It needs to execute the same prepared statement several times with different parameter values.
- The parameter values are not known when the statement is prepared.
- The parameter values need to be converted from one data type to another.

To set a parameter value, an application performs the following operations:

1. Calls **SQLBindParameter** to bind a storage location to a parameter marker and to specify the data types of the storage location and the column associated with the parameter as well as the precision and scale of the parameter
2. Places the value of the parameter in the storage location

These steps can be performed before or after a statement is prepared but must be performed before a statement executes.

Parameter values must be placed in storage locations in the C data types specified in **SQLBindParameter**, shown in the following table.

Parameter Value	SQL Data Type	C Data Type	Stored Value
ABC	SQL_CHAR	SQL_C_CHAR	ABC\0 <sup>a</sup>
10	SQL_INTEGER	SQL_C_SLONG	10
10	SQL_INTEGER	SQL_C_CHAR	10\0 <sup>a</sup>
1 P.M.	SQL_TIME	SQL_C_TIME	13,0,0 <sup>b</sup>
1 P.M.	SQL_TIME	SQL_C_CHAR	{t '13:00:00'}\0 <sup>a,c</sup>

<sup>a</sup> \0 represents a null-termination byte; the null-termination byte is required only if the parameter length is SQL\_NTS.

<sup>b</sup> The numbers in this list are the numbers stored in the fields of the TIME\_STRUCT structure.

<sup>c</sup> The string uses the ODBC date escape clause. For more information, see [“Date, Time, and Time-Stamp Data”](#) on page 6-13.

Storage locations remain bound to parameter markers until the application calls **SQLFreeStmt** with the SQL\_RESET\_PARAMS option or the SQL\_DROP option. An application can bind a different storage area to a parameter marker at any time by calling **SQLBindParameter**. An application can also change the value in a storage location at any time. When a statement is executed, the driver uses the current values in the most recently defined storage locations.



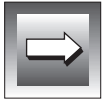
---

## Performing Transactions

In *auto-commit* mode, every SQL statement is a complete transaction that is automatically committed. In *manual-commit* mode, a transaction consists of one or more statements. In manual-commit mode, when an application submits an SQL statement and no transaction is open, the driver implicitly begins a transaction. The transaction remains open until the application commits or rolls back the transaction with **SQLTransact**.

For the Informix driver, the default transaction mode is auto-commit. An application calls **SQLSetConnectOption** to switch between manual-commit and auto-commit mode. If an application switches from manual-commit to auto-commit mode, the driver commits any open transactions on the connection.

Applications should call **SQLTransact** to commit or rollback a transaction, rather than submitting a COMMIT or ROLLBACK statement. The result of a COMMIT or ROLLBACK statement depends on the driver and its associated data source.



***Important:** Committing or rolling back a transaction, either by calling **SQLTransact** or by using the `SQL_AUTOCOMMIT` connection option, can cause the data source to close the cursors and delete the access plans for all hstmts on an hdbc. For more information, see “`SQL_CURSOR_COMMIT_BEHAVIOR`” and “`SQL_CURSOR_ROLLBACK_BEHAVIOR`” on page 13-192.*

## Retrieving Information About the Data-Source Catalog

The following functions, known as catalog functions, return information about a data-source catalog:

- **SQLTables** returns the names of tables stored in a data source.
- **SQLTablePrivileges** returns the privileges associated with one or more tables.
- **SQLColumns** returns the names of columns in one or more tables.
- **SQLColumnPrivileges** returns the privileges associated with each column in a single table.
- **SQLPrimaryKeys** returns the names of columns that comprise the primary key of a single table.

- **SQLSpecialColumns** returns information about the optimal set of columns that uniquely identify a row in a single table or the columns in the table that are automatically updated when any value in the row is updated by a transaction.
- **SQLStatistics** returns statistics about a single table and the indexes associated with the table.
- **SQLProcedures** returns the names of procedures stored in a data source.

Each function returns the information as a result set. Use **SQLBindCol** and **SQLFetch** to retrieve these results.

## Sending Parameter Data at Execution Time

To send parameter data, such as for parameters of the `SQL_LONGVARCHAR` or `SQL_LONGVARBINARY` types when a statement executes, use the following functions:

- **SQLBindParameter**
- **SQLParamData**
- **SQLPutData**

To indicate that your application plans to send parameter data when the statement executes, call **SQLBindParameter** and set the *pcbValue* buffer for the parameter to the result of the `SQL_LEN_DATA_AT_EXEC(length)` macro. If the *fSqlType* argument is `SQL_LONGVARBINARY` or `SQL_LONGVARCHAR` and the driver returns “Y” for the `SQL_NEED_LONG_DATA_LEN` information type in **SQLGetInfo**, *length* is the total number of bytes of data to be sent for the parameter; otherwise, it is ignored.

Set the *rgbValue* argument to a value that, at run time, can be used to retrieve the data. For example, *rgbValue* might point to a storage location that contains the data when the statement executes or to a file that contains the data. The driver returns the value to the application when the statement executes.

When the driver processes a call to **SQLExecute** or **SQLExecDirect** and the executing statement includes a data-at-execution parameter, the driver returns `SQL_NEED_DATA`.

To send the parameter data, an application performs the following operations:

1. Calls **SQLParamData**, which returns *rgbValue* (as set with **SQLBindParameter**) for the first data-at-execution parameter.
2. Calls **SQLPutData** one or more times to send data for the parameter. More than one call is needed if the data value is larger than the buffer; multiple calls are allowed only if the C data type is character or binary and the SQL data type is character, binary, or data-source specific.
3. Calls **SQLParamData** again to indicate that all the data has been sent for the parameter  
If another data-at-execution parameter remains, the driver returns *rgbValue* for that parameter and `SQL_NEED_DATA` for the function return code. Otherwise, it returns `SQL_SUCCESS` for the function return code.
4. Repeats steps 2 and 3 for the remaining data-at-execution parameters.

For additional information, see [“SQLBindParameter” on page 13-23](#).

## Specifying Arrays of Parameter Values

To specify multiple sets of parameter values for a single SQL statement, an application calls **SQLParamOptions**. For example, if there are ten sets of column values to insert into a table, and the same SQL statement can be used for all ten operations, the application can set up an array of values, then submit a single INSERT statement.

If an application uses **SQLParamOptions**, it must allocate enough memory to handle the arrays of values.

## Using ODBC Extensions to SQL

INFORMIX-CLI supports ODBC extensions to SQL as well as Informix-specific extensions. The following section describes the general format of the extensions as defined by the ODBC specification. ODBC defines the following extensions to SQL, which are common to most DBMSs:

- Date, time, and time-stamp data
- Scalar functions, such as numeric and string functions
- Outer joins

The syntax defined by ODBC for these extensions uses the escape clause provided by the X/Open and SQL Access Group SQL CAE specification (1992) to cover vendor-specific extensions to SQL. Its format is as follows:

```
--(*vendor(vendor-name), product(product-name) extension *)--
```

For the ODBC extensions to SQL, *product-name* is always ODBC because the product that defines them is ODBC. *Vendor-name* is always Microsoft because ODBC is a Microsoft product. ODBC also defines a shorthand syntax for these extensions, as follows:

```
{extension}
```

Most DBMSs provide the same extensions to SQL as ODBC does. For this reason, an application might be able to submit an SQL statement using one of these extensions in either of the following ways:

- Use the syntax defined by ODBC  
An application that uses the ODBC syntax is interoperable among DBMSs.
- Use the syntax defined by the DBMS  
An application that uses DBMS-specific syntax is not interoperable among DBMSs.

To determine whether the driver and data source support all the ODBC extensions to SQL, an application calls **SQLGetInfo** with the `SQL_ODBC_SQL_CONFORMANCE` flag. For information about how an application determines whether a specific extension is supported, see [“Supported SQL” on page 13-188](#) and [“SQL Limits” on page 13-189](#).



**Important:** Informix database servers provide extensions to SQL other than those defined by ODBC. To use one of these extensions, refer to [Informix Guide to SQL: Syntax](#). If your application uses the Informix-specific syntax, it is not interoperable among DBMSs.

### ***Date, Time, and Time-Stamp Data***

The escape clauses that ODBC uses for date, time, and time-stamp data are shown in the following example:

```
--(*vendor(Microsoft),product(ODBC) d 'value' *)--
--(*vendor(Microsoft),product(ODBC) t 'value' *)--
--(*vendor(Microsoft),product(ODBC) ts 'value' *)--
```

- d** indicates value is a date in the “yyyy-mm-dd” format
- t** indicates value is a time in the “hh:mm:ss” format
- ts** indicates value is a time stamp in the “yyyy-mm-dd hh:mm:ss[.f...]” format

The shorthand syntax for date, time, and time-stamp data is as follows:

```
{d 'value'}
{t 'value'}
{ts 'value'}
```

For example, each of the following statements updates the birthday of Walter Jacobs in the **EMPLOYEE** table. The first statement uses the escape-clause syntax. The second statement uses the shorthand syntax. The third statement uses the native syntax for Informix for a DATE column and is not interoperable among DBMSs.

```
UPDATE EMPLOYEE
  SET BIRTHDAY=--(*vendor(Microsoft),product(ODBC) d
    '1938-01-15' *)--
  WHERE NAME='Jacobs, Walter'

UPDATE EMPLOYEE
  SET BIRTHDAY={d '1938-01-15'}
  WHERE NAME='Jacobs, Walter'

UPDATE EMPLOYEE
  SET BIRTHDAY='01/15/1938'
  WHERE NAME='Jacobs, Walter'
```

The ODBC escape clauses for date, time, and timestamp literals can be used in parameters with a C data type of `SQL_C_CHAR`. For example, the following statement uses a parameter to update the birthday of Jane Smith in the `EMPLOYEE` table:

```
UPDATE EMPLOYEE SET BIRTHDAY=? WHERE NAME='Smith, Jane'
```

A storage location of type `SQL_C_CHAR` bound to the parameter might contain any of the following values. The first value uses the escape clause syntax. The second value uses the shorthand syntax. The third value uses the native syntax for Informix for a `DATE` column and is not interoperable among DBMSs.

```
--(*vendor(Microsoft),product(ODBC) d '1938-01-15' *)--  
"{d '1938-01-15'}"  
"01-15-1938"
```

An application can also send date, time, or time-stamp values as parameters using the C structures defined by the C data types `SQL_C_DATE`, `SQL_C_TIME`, and `SQL_C_TIMESTAMP`.

To determine if a data source supports date, time, or time-stamp data, an application calls `SQLGetTypeInfo`.

### ***Scalar Functions***

Scalar functions, such as string length, absolute value, or current date, can be used on columns of a result set and on columns that restrict rows of a result set. The escape clause that ODBC uses for scalar functions is shown in the following example:

```
--(*vendor(Microsoft),product(ODBC) fn scalar-function *)--
```

The shorthand syntax for scalar functions is as follows:

```
{fn scalar-function}
```

For example, each of the following statements creates the same result set of uppercase employee names. The first statement uses the escape-clause syntax. The second statement uses the shorthand syntax. The third statement uses the native syntax for Informix (for UNIX) and is not interoperable among DBMSs.

```
SELECT --(*vendor(Microsoft),product(ODBC) fn
LENGTH(NAME)*)--
    FROM EMPLOYEE

SELECT {fn LENGTH(NAME)} FROM EMPLOYEE

SELECT length(NAME) FROM EMPLOYEE
```

To determine which scalar functions are supported by a data source, an application calls **SQLGetInfo** with the `SQL_NUMERIC_FUNCTIONS`, `SQL_STRING_FUNCTIONS`, `SQL_SYSTEM_FUNCTIONS`, and `SQL_TIMEDATE_FUNCTIONS` flags. For more information on scalar functions that Informix database servers support, refer to [Informix Guide to SQL: Syntax](#).

### ***LIKE Predicate Escape Characters***

In a LIKE predicate, the percent sign (%) matches zero or more of any character and the underscore character (\_) matches any character. The percent and underscore characters can be used as literals in a LIKE predicate by preceding them with an escape character. The escape clause that ODBC uses to define the LIKE predicate escape character is shown in the following example:

```
--(*vendor (Microsoft), product (ODBC) escape 'escape-
character' *)--
```

In this example, *escape-character* is any character supported by the data source. The shorthand syntax for the LIKE predicate escape character is as follows:

```
{escape 'escape-character' }
```

For example, each of the following statements creates the same result set of department names that start with the characters '%AAA.' The first statement uses the escape clause syntax. The second statement uses the shorthand syntax. The third statement uses the native syntax for Informix (for UNIX) and is not interoperable among DBMSs. The second percent character (%) in each LIKE predicate is a wild-card character that matches zero or more of any character.

```
SELECT NAME FROM DEPT WHERE NAME LIKE '\%AAA%' --
(*vendor(Microsoft),product(ODBC) escape '\%*')--

SELECT NAME FROM DEPT WHERE NAME LIKE '\%AAA%' {escape '\'}

SELECT NAME FROM DEPT WHERE NAME LIKE '\%AAA%' ESCAPE '\'
```

To determine whether a data source supports LIKE predicate escape characters, an application calls **SQLGetInfo** with the information type **SQL\_LIKE\_ESCAPE\_CLAUSE**.

### **Outer Joins**

ODBC supports the ANSI SQL-92 LEFT OUTER JOIN syntax. The escape clause that ODBC uses for outer joins is shown in the following example:

```
--(*vendor(Microsoft),product(ODBC) oj outer-join *)--
```

**Outer-join** is used as follows:

```
table-reference LEFT OUTER JOIN {table-reference | outer-join}
ON search-condition
```

Also, in the previous example, *table-reference* specifies a table name, and *search-condition* specifies the join condition between the table references. The shorthand syntax for outer joins is as follows:

```
{oj outer-join}
```

An outer-join request must appear after the FROM keyword and before the WHERE clause if one exists. For complete syntax information, see [Appendix B](#).



For example, each of the following statements creates the same result set of the names and departments of employees working on project 544. The first statement uses the escape clause syntax. The second statement uses the shorthand syntax. The third statement uses the native syntax for Informix (for UNIX) and is not interoperable among DBMSs.

```
SELECT EMPLOYEE.NAME, DEPT.DEPTNAME
FROM --(*vendor(Microsoft),product(ODBC) oj
EMPLOYEE LEFT OUTER JOIN DEPT ON EMPLOYEE.DEPTID=DEPT.DEPTID*)--
WHERE EMPLOYEE.PROJID=544
```

```
SELECT EMPLOYEE.NAME, DEPT.DEPTNAME
FROM {oj EMPLOYEE LEFT OUTER JOIN DEPT
ON EMPLOYEE.DEPTID=DEPT.DEPTID}
WHERE EMPLOYEE.PROJID=544
```

```
SELECT EMPLOYEE.NAME, DEPT.DEPTNAME
FROM EMPLOYEE, OUTER DEPT
WHERE (EMPLOYEE.PROJID=544) AND (EMPLOYEE.DEPTID = DEPT.DEPTID)
```

To determine the level of outer joins that a data-source supports, an application calls **SQLGetInfo** with the `SQL_OUTER_JOINS` flag. Data sources can support two-table outer joins, partially support multitable outer joins, fully support multitable outer joins, or not support outer joins.

## Procedures

An application can call a procedure in place of an SQL statement. The escape clause that ODBC uses for calling a procedure is as follows:

```
--(*vendor(Microsoft),product(ODBC)
[?]= call procedure-name[([parameter][,[parameter]]...)] *)--
```

In the example, *procedure-name* specifies the name of a procedure stored on the data source and *parameter* specifies a procedure parameter. A procedure can have zero or more parameters and can return a value. The shorthand syntax for procedure invocation is as follows:

```
{[?]=call procedure-name[([parameter][,[parameter]]...)]}
```

For output parameters, *parameter* must be a parameter marker. For input and input/output parameters, *parameter* can be a literal, a parameter marker, or not specified. If *parameter* is a literal or is not specified for an input/output parameter, the driver discards the output value. If *parameter* is not specified for an input or input/output parameter, the procedure uses the default value of the parameter as the input value; the procedure also uses the default value if *parameter* is a parameter marker and the *pcbValue* argument in **SQLBindParameter** is `SQL_DEFAULT_PARAM`. If a procedure call includes parameter markers (including the “?” parameter marker for the return value), the application must bind each marker by calling **SQLBindParameter** prior to calling the procedure.



**Important:** For some data sources, *parameter* cannot be a literal value. For all data sources, it can be a parameter marker. For maximum interoperability, applications should always use a parameter marker for *parameter*.

If an application specifies a return value parameter for a procedure that does not return a value, the driver sets the *pcbValue* buffer specified in **SQLBindParameter** for the parameter to `SQL_NULL_DATA`. If the application omits the return value parameter for a procedure which returns a value, the driver ignores the value returned by the procedure.

If a procedure returns a result set, the application retrieves the data in the result set in the same manner as it retrieves data from any other result set.

For example, each of the following statements uses the procedure `EMPS_IN_PROJ` to create the same result set of names of employees working on a project. The first statement uses the escape-clause syntax. The second statement uses the shorthand syntax.

```
--(*vendor(Microsoft),product(ODBC) call EMPS_IN_PROJ(?)*)--  
{call EMPS_IN_PROJ(?)}
```

To determine if a data source supports procedures, an application calls **SQLGetInfo** with the `SQL_PROCEDURES` information type. To retrieve a list of the procedures stored in a data source, an application calls **SQLProcedures**.

## Related Functions

INFORMIX-CLI also provides the following functions related to SQL statements. For more information about these functions, see [Chapter 13, “INFORMIX-CLI Function Reference.”](#)

Function	Description
<b>SQLNativeSql</b>	Retrieves the SQL statement as processed by the data source, with escape sequences translated to SQL code used by the data source.
<b>SQLNumParams</b>	Retrieves the number of parameters in an SQL statement.
<b>SQLSetStmtOption</b> <b>SQLSetConnectOption</b> <b>SQLGetStmtOption</b>	Sets or retrieves statement options, such as orientation for binding rowsets, maximum amount of variable-length data to return, maximum number of result set rows to return, and query time-out value. <b>SQLSetConnectOption</b> sets options for all the statements in a connection.



---

# Retrieving Results

Assigning Storage for Results (Binding) . . . . .	7-4
Determining the Characteristics of a Result Set . . . . .	7-4
Fetching Result Data . . . . .	7-5
Using Cursors . . . . .	7-6
Retrieving Data from Unbound Columns . . . . .	7-7
Assigning Storage for Rowsets (Binding) . . . . .	7-7
Column-Wise and Row-Wise Binding . . . . .	7-7
Retrieving Rowset Data . . . . .	7-9
Using Block and Scrollable Cursors . . . . .	7-9
Block Cursors . . . . .	7-9
Scrollable Cursors . . . . .	7-10
Specifying the Cursor Type . . . . .	7-12
Specifying Cursor Concurrency . . . . .	7-12
Modifying Result Set Data with Positioned Update and Delete Statements . . . . .	7-13
Processing Multiple Results . . . . .	7-15



**A** **SELECT** statement is used to retrieve data that meets a given set of specifications. In the following example, a **SELECT** statement retrieves all columns for employees named Jones from all the rows in the **EMPLOYEE** table.

```
SELECT * FROM EMPLOYEE WHERE EMPNAME = "Jones"
```

**INFORMIX-CLI** functions can also retrieve data. For example, **SQLColumns** retrieves data about columns in the data source. These sets of data, called result sets, can contain zero or more rows.

Other **SQL** statements, such as **GRANT** or **REVOKE**, do not return result sets. For these statements, the return code from **SQLExecute** or **SQLExecDirect** is usually the only source of information about whether the statement is successful. For **INSERT**, **UPDATE**, and **DELETE** statements, an application can call **SQLRowCount** to return the number of affected rows.

The steps an application takes to process a result set depends on what is known about it. The following list defines known and unknown result sets:

- |                    |  |
|--------------------|--|
| Known result set   | The application knows the exact form of the <b>SQL</b> statement, and therefore the result set, when the statement compiles. For example, the query <b>SELECT EMPNO, EMPNAME FROM EMPLOYEE</b> returns two specific columns.   |
| Unknown result set | The application does not know the exact form of the <b>SQL</b> statement, and therefore the result set, when the statement compiles. For example, the ad hoc query <b>SELECT * FROM EMPLOYEE</b> returns all the currently defined columns in the <b>EMPLOYEE</b> table. The application might not predict the format of these results before it executes the command. |

---

## Assigning Storage for Results (Binding)

An application can assign storage for results before or after it executes an SQL statement. If an application prepares or executes the SQL statement first, it can inquire about the result set before it assigns storage for results. For example, if the result set is unknown, the application must retrieve the number of columns in the result set before it can assign storage for them.

To associate storage for a column of data, an application calls **SQLBindCol** and passes it the following information:

- The data type to which the data is to be converted  
For more information, see [Appendix C](#).
- The address of an output buffer for the data  
The application must allocate this buffer, which must be sufficient to hold the data in its converted form.
- The length of the output buffer  
This value is ignored if the returned data has a fixed width in C, such as an integer, real number, or date structure.
- The address of a storage buffer in which to return the number of bytes of available data

---

## Determining the Characteristics of a Result Set

To determine the characteristics of a result set, an application can perform the following actions:

- Call **SQLNumResultCols** to determine how many columns a request returned
- Call **SQLColAttributes** or **SQLDescribeCol** to describe a column in the result set

If the result set is unknown, an application can use the information from these functions to bind the columns in the result set. An application can call these functions at any time after a statement is prepared or executed.





**Important:** Although **SQLRowCount** can sometimes return the number of rows in a result set, it is not guaranteed to do so. Few data sources support this functionality and interoperable applications should not rely on it.

**Tip:** For optimal performance, an application should call **SQLColAttributes**, **SQLDescribeCol**, and **SQLNumResultCols** after a statement executes. In data sources that emulate statement preparation, these functions sometimes execute more slowly before a statement executes because the information returned by them is not readily available until after the statement has executed.

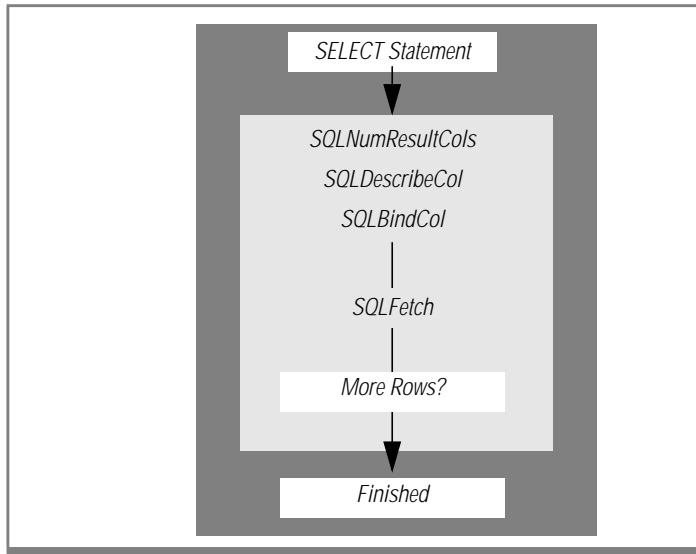
---

## Fetching Result Data

You can perform several operations to retrieve data once the characteristics of the result set are known. To retrieve a row of data from the result set, an application performs the following operations:

1. Calls **SQLBindCol** to bind the columns of the result set to storage locations if it has not already done so.
2. Calls **SQLFetch** to move to the next row in the result set and retrieve data for all bound columns.

Figure 7-1 shows the operations an application uses to retrieve data from the result set.



**Figure 7-1**  
Operations That an  
Application Uses to  
Retrieve Data from  
the Result Set

## Using Cursors

A driver maintains a cursor to indicate the current position in the result set, much as the cursor on a CRT screen indicates the current position.

You can use **SQLFetch** or **SQLExtendedFetch** to retrieve data from a result set. Each time an application calls **SQLFetch**, the driver moves the cursor to the next row and returns that row. To retrieve a row of data that has already been retrieved from the result set, the application must close the cursor by calling **SQLFreeStmt** with the **SQL\_CLOSE** option, reexecute the **SELECT** statement, and fetch rows with **SQLFetch** until the target row is retrieved.



**Important:** *Committing or rolling back a transaction, either by calling **SQLTransact** or by using the **SQL\_AUTOCOMMIT** connection option, can cause the data source to close the cursors for all hstmts on an hdbc.*

For more information, see “**SQL\_CURSOR\_COMMIT\_BEHAVIOR**” and “**SQL\_CURSOR\_ROLLBACK\_BEHAVIOR**” on page 13-192.

## Retrieving Data from Unbound Columns

To retrieve data from unbound columns (that is, columns for which storage has not been assigned with **SQLBindCol**), an application uses **SQLGetData**. The application first calls **SQLFetch** or **SQLExtendedFetch** to position the cursor on the next row. It then calls **SQLGetData** to retrieve data from specific unbound columns.

An application can retrieve data from bound and unbound columns in the same row. It calls **SQLBindCol** to bind as many columns as you specify. It calls **SQLFetch** or **SQLExtendedFetch** to position the cursor on the next row of the result set and to retrieve all bound columns. It then calls **SQLGetData** to retrieve data from unbound columns.

If the column data type is character, binary, or data-source specific and the column contains more data than can be retrieved in a single call, an application might call **SQLGetData** more than once for that column, as long as the data is being transferred to a buffer of type **SQL\_C\_CHAR** or **SQL\_C\_BINARY**. For example, data of the **SQL\_LONGVARBINARY** and **SQL\_LONGVARCHAR** types might need to be retrieved in several parts.

The Informix driver can return data with **SQLGetData** for both bound and unbound columns in any order. For maximum interoperability, however, your application should call **SQLGetData** only for columns to the right of the rightmost bound column and then only in left-to-right order.

## Assigning Storage for Rowsets (Binding)

In addition to binding individual rows of data, an application can call **SQLBindCol** to assign storage for a *rowset* (one or more rows of data).

To specify how many rows of data are in a rowset, an application calls **SQLSetStmtOption** with the **SQL\_ROWSET\_SIZE** option.

### *Column-Wise and Row-Wise Binding*

By default, rowsets are bound in column-wise fashion. They can also be bound in row-wise fashion.

To assign storage for column-Cwise bound results, an application performs the following operations:

1. Allocates an array of data storage buffers  
The array has as many elements as there are rows in the rowset.
2. Allocates an array of storage buffers to hold the number of bytes that are available to return for each data value  
The array has as many elements as there are rows in the rowset.
3. Calls **SQLBindCol** and specifies the address of the data array, the size of one element of the data array, the address of the number-of-bytes array, and the type to which the data will be converted  
When data is retrieved, the driver will use the array element size to determine where to store successive rows of data in the array.

To assign storage for row-wise bound results, an application performs the following operations:

1. Declares a structure that can hold a single row of retrieved data and the associated data lengths  
To bind each column, the structure contains one field to contain data and one field to contain the number of bytes of data that are available to return.
2. Allocates an array of these structures  
This array has as many elements as there are rows in the rowset.
3. Calls **SQLBindCol** for each column to be bound.
4. In each call, the application specifies the address of the column data field in the first array element, the size of the data field, the address of the column number-of-bytes field in the first array element, and the type to which the data will be converted.
5. Calls **SQLSetStmtOption** with the `SQL_BIND_TYPE` option and specifies the size of the structure  
When the data is retrieved, the driver uses the structure size to determine where to store successive rows of data in the array.

## Retrieving Rowset Data

Before it retrieves rowset data, an application calls **SQLSetStmtOption** with the `SQL_ROWSET_SIZE` option to specify the number of rows in the rowset. It then binds columns in the rowset with **SQLBindCol**. The rowset is bound using a column- or row-wise method. For more information, see [“Assigning Storage for Rowsets \(Binding\)” on page 7-7](#).

To retrieve rowset data, an application calls **SQLExtendedFetch**.

For maximum interoperability, an application should not use **SQLGetData** to retrieve data from unbound columns in a block of data (more than one row) that has been retrieved with **SQLExtendedFetch**. To determine if a driver can return data with **SQLGetData** from a block of data, an application calls **SQLGetInfo** with the `SQL_GETDATA_EXTENSIONS` option.

## Using Block and Scrollable Cursors

Cursors in SQL scroll forward through a result set, returning one row at a time. However, interactive applications often require forward and backward scrolling, absolute or relative positioning within the result set, and the ability to retrieve and update blocks of data, or *rowsets*.

A *block* cursor attribute allows an application to retrieve and update rowset data. A *scrollable* cursor attribute allows an application to scroll forward or backward through the result set or move to an absolute or relative position in the result set. Cursors can have one or both attributes.

### ***Block Cursors***

An application calls **SQLSetStmtOption** with the `SQL_ROWSET_SIZE` option to specify the rowset size. The application can call **SQLSetStmtOption** to change the rowset size at any time. Each time the application calls **SQLExtendedFetch**, the driver returns the next *rowset-size* rows of data. After the data is returned, the cursor points to the first row in the rowset; by default, the rowset size is one.

## **Scrollable Cursors**

Applications have different needs to sense changes in the tables underlying a result set. For example, when balancing financial data, an accountant needs data that appears static; it is impossible to balance books when the data is continually changing. When selling concert tickets, a clerk needs up-to-the-minute, or dynamic, data on which tickets are still available. Various cursor models are designed to meet these needs, each requiring different sensitivities to changes in the tables that underlie the result set.

### *Static Cursors*

In *static* cursors, the data in the underlying tables appears to be static. The membership, order, and values in the result set used by a static cursor are generally fixed when the cursor is opened. Rows updated, deleted, or inserted by other users, including other cursors in the same application, are not detected by the cursor until it closes and then reopens; the `SQL_STATIC_SENSITIVITY` information type returns whether the cursor can detect rows it has updated, deleted, or inserted.

Static cursors are commonly implemented by taking a snapshot of the data or locking the result set. In the former case, the cursor diverges from the underlying tables as other users make changes; in the latter case, other applications and cursors in the same application cannot change the data.

### *Dynamic Cursors*

In *dynamic* cursors, the data appears to be dynamic. The membership, order, and values in the result set used by a dynamic cursor are always changing. Rows updated, deleted, or inserted by all users (the cursor, other cursors in the same application, and other applications) are detected by the cursor when data is fetched. Although dynamic cursors are ideal for many situations, they are difficult to implement.

### *Keyset-Driven Cursors*

*Keyset-driven* cursors possess some attributes of static and dynamic cursors. Like static cursors, the membership and ordering of the result set of a keyset-driven cursor is generally fixed when the cursor opens. As with dynamic cursors, most changes to the values in the underlying result set are visible to the cursor when data is fetched.

When a keyset-driven cursor opens, the driver saves the keys for the entire result set, which fixes the membership and order of the result set. As the cursor scrolls through the result set, the driver uses the keys in this *keyset* to retrieve the current data values for each row in the rowset. Because data values are retrieved only when the cursor scrolls to a given row, updates to that row by other users (including other cursors in the same application) after the cursor was opened are visible to the cursor.

If the cursor scrolls to a row of data that has been deleted by other users (including other cursors in the same application), the row appears as a *hole* in the result set because the key is still in the keyset, but the row is no longer in the result set. Updating the key values in a row is considered to be deleting the existing row and inserting a new row; therefore, rows of data for which the key values have been changed also appear as holes. When the driver encounters a hole in the result set, it returns a status code for the row of `SQL_ROW_DELETED`.

Rows of data inserted into the result set by other users (including other cursors in the same application) after the cursor was opened are not visible to the cursor because the keys for those rows are not in the keyset.

The `SQL_STATIC_SENSITIVITY` information type returns whether the cursor can detect rows it has deleted or inserted. Because updating key values in a keyset-driven cursor is considered to be deleting the existing row and inserting a new row, keyset-driven cursors can always detect rows that they have updated.

### *Mixed (Keyset/Dynamic) Cursors*

If a result set is large, it might be impractical for the driver to save the keys for the entire result set. Instead, the application can use a *mixed* cursor. In a mixed cursor, the keyset is smaller than the result set, but larger than the rowset.

Within the boundaries of the keyset, a mixed cursor is keyset driven; that is, the driver uses keys to retrieve the current data values for each row in the rowset. When a mixed cursor scrolls beyond the boundaries of the keyset, it becomes dynamic, so the driver simply retrieves the next *rowset-size* rows of data. The driver then constructs a new keyset, which contains the new rowset.

For example, assume a result set has 1000 rows and uses a mixed cursor with a keyset size of 100 and a rowset size of 10. When the cursor opens, the driver (depending on the implementation) saves keys for the first 100 rows and retrieves data for the first 10 rows. If another user deletes row 11 and the cursor then scrolls to row 11, the cursor detects a hole in the result set; the key for row 11 is in the keyset, but the data is no longer in the result set. A keyset-driven cursor behaves the same way. However, if another user deletes row 101 and the cursor then scrolls to row 101, the cursor does not detect a hole; the key for the row 101 is not in the keyset. Instead, the cursor retrieves the data for the row that was originally row 102. A dynamic cursor behaves the same way.

### ***Specifying the Cursor Type***

To specify the cursor type, an application calls **SQLSetStmtOption** with the `SQL_CURSOR_TYPE` option. The application can specify a cursor that scrolls forward, a static cursor, a dynamic cursor, a keyset-driven cursor, or a mixed cursor. If the application specifies a mixed cursor, it also specifies the size of the keyset that the cursor uses.

To use the ODBC cursor library, an application calls **SQLSetConnectOption** with the `SQL_ODBC_CURSORS` option before it connects to the data source. For more information on the ODBC cursor library, see the *Microsoft ODBC Programmer's Reference and SDK Guide, Version 2.0*.

An application calls **SQLExtendedFetch** to scroll the cursor backward, forward, or to an absolute or relative position in the result set.

### ***Specifying Cursor Concurrency***

*Concurrency* is the ability of more than one user to use the same data at the same time. A transaction is *serializable* if it is performed in a manner in which it appears as if no other transactions operate on the same data at the same time. For example, assume one transaction doubles data values and another adds 1 to data values. If the transactions are serializable and both attempt to operate on the values 0 and 10 at the same time, the final values are 1 and 21 or 2 and 22, depending on which transaction occurs first. If the transactions are not serializable, the final values are 1 and 21, 2 and 22, 1 and 22, or 2 and 21; the sets of values 1 and 22, and 2 and 21 are the result of the transactions acting on each value in a different order.



Serializability is necessary to maintain database integrity. For cursors, it is implemented at the expense of concurrency by locking the result set. A compromise between serializability and concurrency is *optimistic concurrency control*. In a cursor that uses optimistic concurrency control, the driver does not lock rows when it retrieves them. When the application requests an update or delete operation, the driver or data source checks if the row has changed. If the row is unchanged, the driver or data source prevents other transactions from changing the row until the operation is complete. If the row has changed, the transaction that contains the update or delete operation fails.

To specify the concurrency used by a cursor, an application calls **SQLSetStmtOption** with the `SQL_CONCURRENCY` option. The application can specify that the cursor is read-only, locks the result set, uses optimistic concurrency control and compares row versions to determine if a row has changed, or uses optimistic concurrency control and compares data values to determine if a row has changed.

## Modifying Result Set Data with Positioned Update and Delete Statements

To modify data in the result set, an application can update or delete the row to which the cursor currently points. This is known as a positioned update or delete statement

INFORMIX-CLI positioned update and delete statements are similar to such statements in embedded SQL. After executing a `SELECT` statement to create a result set, an application calls **SQLFetch** one or more times to position the cursor on the row to be updated or deleted. To update or delete the row, the application then executes an SQL statement with the following syntax on a different *hstmt*:

```
UPDATE table-name
SET column-identifier = {expression | NULL}
[, column-identifier = {expression | NULL}]...
WHERE CURRENT OF cursor-name
DELETE FROM table-name WHERE CURRENT OF cursor-name
```

Positioned update and delete statements require cursor names. An application can name a cursor with **SQLSetCursorName**. If the application has not named the cursor by the time the driver executes a **SELECT** statement, the driver generates a cursor name. To retrieve the cursor name for an *hstmt*, an application calls **SQLGetCursorName**.

To execute a positioned update or delete statement, an application must follow these guidelines:

- The **SELECT** statement that creates the result set must use a **FOR UPDATE** clause.
- The cursor name used in the **UPDATE** or **DELETE** statement must be the same as the cursor name associated with the **SELECT** statement.
- The application must use different *hstmts* for the **SELECT** statement and the **UPDATE** or **DELETE** statement.
- The *hstmts* for the **SELECT** statement and the **UPDATE** or **DELETE** statement must be on the same connection.

To determine if a data source supports positioned update and delete statements, an application calls **SQLGetInfo** with the **SQL\_POSITIONED\_STATEMENTS** option.

In ODBC 1.0, positioned update, positioned delete, and **SELECT FOR UPDATE** statements were part of the core SQL grammar. In ODBC 2.x, positioned update, positioned delete, and **SELECT FOR UPDATE** statements are part of the extended grammar. Applications that use the SQL conformance level to determine whether these statements are supported also need to check the version number of the driver to interpret the information correctly. In particular, applications that use these features with ODBC 1.0 drivers need to check explicitly for these capabilities in ODBC 2.x drivers.

## Processing Multiple Results

SELECT statements return result sets. UPDATE, INSERT, and DELETE statements return a count of affected rows. If any of these statements are batched, submitted with arrays of parameters, or in procedures, they can return multiple result sets or counts.

To process a batch of statements, statement with arrays of parameters, or procedure that returns multiple result sets or row counts, an application performs the following operations:

1. Calls **SQLExecute** or **SQLExecDirect** to execute the statement or procedure
2. Calls **SQLRowCount** to determine the number of rows affected by an UPDATE, INSERT, or DELETE statement  
For statements or procedures that return result sets, the application calls functions to determine the characteristics of the result set and retrieve data from the result set.
3. Calls **SQLMoreResults** to determine if another result set or row count is available.
4. Repeats steps 2 and 3 until **SQLMoreResults** returns **SQL\_NO\_DATA\_FOUND**.



---

# Retrieving Status and Error Information

Function Return Codes . . . . .	8-3
Retrieving Error Messages . . . . .	8-4
Error Messages . . . . .	8-5
Error Text Format . . . . .	8-5
Sample Error Messages . . . . .	8-6
Sample Error Returned from the Driver . . . . .	8-6
Sample Error Returned from the Driver Manager. . . . .	8-7
Sample Error Returned from the Data Source . . . . .	8-7
Processing Error Messages . . . . .	8-8



**T**his chapter defines INFORMIX-CLI return codes and error-handling protocol. The return codes indicate whether a function succeeded, succeeded but returned a warning, or failed. The error-handling protocol defines how the components in an INFORMIX-CLI connection construct and return error messages through **SQLError**.

The protocol describes the following points:

- Use of the error text to identify the source of an error
- Rules to ensure consistent and useful error information
- Responsibility for setting the SQLSTATE based on the native error

---

## Function Return Codes

When an INFORMIX-CLI application calls a function, the driver executes the function and returns a predefined code. These return codes indicate success, warning, or failure status. The following table defines the return codes.

Return Code	Description
SQL_SUCCESS	Function completed successfully; no additional information is available.
SQL_SUCCESS_WITH_INFO	Function completed successfully, possibly with a nonfatal error. The application can call <b>SQLError</b> to retrieve additional information.
SQL_NO_DATA_FOUND	All rows from the result set have been fetched.

(1 of 2)

Return Code	Description
SQL_ERROR	Function failed. The application can call <b>SQLError</b> to retrieve error information.
SQL_INVALID_HANDLE	Function failed due to an invalid environment handle, connection handle, or statement handle. This situation indicates a programming error. No additional information is available from <b>SQLError</b> .
SQL_NEED_DATA	While the driver was processing a statement, it determined that the application needs to send parameter data values.

(2 of 2)

The application is responsible for taking the appropriate action based on the return code.

---

## Retrieving Error Messages

If a function other than **SQLError** returns `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`, an application can call **SQLError** to obtain additional information. The application might need to call **SQLError** more than once to retrieve all the error messages from a function because a function might return more than one error message. When the application calls a different function, the error messages from the previous function are deleted.

Additional error or status information can come from one of the following sources:

- Error or status information from an ODBC function, indicating that a programming error was detected
- Error or status information from the data source, indicating that an error occurred during SQL statement processing

The information returned by **SQLError** is in the same format as that provided by `SQLSTATE` in the X/Open and SQL Access Group SQL CAE specification (1992). **SQLError** never returns error information about itself.



---

## Error Messages

ODBC defines a layered architecture to connect an application to a data source. In INFORMIX-CLI, **SQLERROR** returns error messages that follow the standard ODBC format. Error messages must not only explain the error but also provide the identity of the component in which it occurred. Because **SQLERROR** does not return the identity of the component in which the error occurred, this information must be embedded in the error text.

### Error Text Format

Error messages returned by **SQLERROR** come from two sources: data sources and ODBC components (the driver manager or the driver). Consequently, the error text returned by **SQLERROR** has two formats: one for errors that occur in a data source and one for errors that occur in ODBC components.

If an ODBC component receives an error message from a data source, it must identify the data source as the source of the error. It must also identify itself as the component that received the error. For errors that occur in a data source, the error text must use the following format::

```
[vendor_identifier][ODBC_component_identifier]  
[data_source_identifier]data_source_supplied_text
```

If the source of an error is an ODBC component, the error message must explain which component. For errors that do not occur in a data source, the error text must use the following format:

```
[vendor_dentifier][ODBC_component_identifier]  
component_supplied_text
```

The following table shows the meaning of each element.

Element	Meaning
<i>vendor_identifier</i>	Identifies the vendor of the component in which the error occurred or that received the error directly from the data source.
<i>ODBC_component_identifier</i>	Identifies the component in which the error occurred or that received the error directly from the data source.
<i>data_source_identifier</i>	Identifies the data source. For multiple-tier drivers, this is the DBMS product.
<i>component_supplied_text</i>	Error text generated by the ODBC component. (the driver manager or the driver).
<i>data_source_supplied_text</i>	Error text generated by the data source

The brackets ( [ ] ) are included in the error text; they do not indicate optional items.

## Sample Error Messages

The following examples show how various components in a connection might generate the text of error messages and how INFORMIX-CLI might return the error messages to the application with **SQLERROR**.

### *Sample Error Returned from the Driver*

An INFORMIX-CLI driver sends requests to a DBMS and returns information to the application through the driver manager. Because it is the component that interfaces with the driver manager, it formats and returns arguments for **SQLERROR**.

For example, if an INFORMIX-CLI driver for INFORMIX-SE encounters a duplicate cursor name, it might return the following arguments for **SQLERROR**:

```
szSQLState = "3C000"  
pfNativeError = NULL  
szErrorMsg = "[Informix][ODBC Informix Driver]  
Duplicate cursor name:EMPLOYEE_CURSOR."  
pcbErrorMsg = 67
```

Because the error occurred in the driver, it adds prefixes to the error text for the vendor (Informix) and the driver (ODBC Informix Driver).

### ***Sample Error Returned from the Driver Manager***

The driver manager can also generate error messages. For example, if an application passed an invalid argument value to **SQLDataSources**, the driver manager might format and return the following arguments for **SQLERROR**:

```
szSQLState = "S1009"  
pfNativeError = NULL  
szErrorMsg = "[Informix][ODBC DLL]Invalid argument value:SQLDataSources."  
pcbErrorMsg = 60
```

Because the error occurred in the driver manager, it adds prefixes to the error text for its vendor (Informix) and its identifier (ODBC DLL).

### ***Sample Error Returned from the Data Source***

If the DBMS could not find the table **EMPLOYEE**, the driver might format and return the following arguments for **SQLERROR**:

```
szSQLState = "S0002"  
pfNativeError = -206  
szErrorMsg = "[Informix][ODBC Informix Driver][Informix]The  
specified table (EMPLOYEE) is not in the database."  
pcbErrorMsg = 96
```

Because the error occurred in the data source, the driver added a prefix for the data source identifier (Informix) to the error text. Because the driver component interfaced with the data source, it adds prefixes for its vendor (Informix) and identifier (ODBC Informix Driver) to the error text. For a description of an error that the data source returns, look up the native error value in the [Informix Error Messages](#) manual.

---

## **Processing Error Messages**

In your application, provide users with all the error information available through **SQLError**: the `SQLSTATE`, the native error code, the error text, and the source of the error. The application might parse the error text to separate the text from the information that identifies the source of the error. The application must take appropriate action based on the error or provide the user with a choice of actions.

---

# Terminating Transactions and Connections

Terminating Statement Processing . . . . .	9-3
Terminating Transactions. . . . .	9-4
Terminating Connections. . . . .	9-4



# T

he INFORMIX-CLI interface provides functions that terminate statements, transactions, and connections, as well as free statement (*hstmt*), connection (*hdbc*), and environment (*henv*) handles.

---

## Terminating Statement Processing

To free the resources associated with a statement handle, an application calls **SQLFreeStmt**. The **SQLFreeStmt** function has the following options:

- **SQL\_CLOSE**  
This option closes the cursor, if one exists, and discards pending results. The application can use the statement handle again later.
- **SQL\_DROP**  
This option closes the cursor, if one exists, discards pending results, and frees all resources associated with the statement handle.
- **SQL\_UNBIND**  
This option frees all return buffers bound by **SQLBindCol** for the statement handle.
- **SQL\_RESET\_PARAMS**  
This option frees all parameter buffers requested by **SQLBindParameter** for the statement handle.

---

## Terminating Transactions

An application calls **SQLTransact** to commit or roll back the current transaction.

---

## Terminating Connections

An application terminates a connection to a driver or to a data source by closing the connection associated with a connection handle and freeing the connection and environment handles. To terminate a connection to a driver and data source, an application performs the following operations:

1. Calls **SQLDisconnect** to close the connection  
You can then use the handle to reconnect to the same data source or to a different data source.
2. Calls **SQLFreeConnect** to free the connection handle and free all resources associated with the handle
3. Calls **SQLFreeEnv** to free the environment handle and free all resources associated with the handle



---

# Constructing an INFORMIX-CLI Application

Static SQL Example . . . . .	10-4
Interactive Ad-Hoc Query Example . . . . .	10-7



**T**

his chapter provides the following examples of C-language source code for INFORMIX-CLI-enabled applications:

- An example that uses static SQL functions to create a table, add data to it, and select the inserted data
- An example of interactive, ad-hoc query processing

---

## Static SQL Example

The following example constructs SQL statements within the application. The example comments include equivalent embedded SQL calls for illustrative purposes.

```
#include "sql.h"
#include <string.h>

#ifdef NULL
#define NULL 0
#endif
#define MAX_NAME_LEN 50
#define MAX_STMT_LEN 100
int print_err(HDBC hdbc, HSTMT hstmt);

int example1(server, uid, pwd)
    UCHAR * server;
    UCHAR * uid;
    UCHAR * pwd;
    {
    HENV henv;
    HDBC hdbc;
    HSTMT hstmt;

    SDWORD id;
    UCHAR name[MAX_NAME_LEN + 1];
    UCHAR create[MAX_STMT_LEN]
    UCHAR insert[MAX_STMT_LEN]
    UCHAR select[MAX_STMT_LEN]
    SDWORD namelen;
    RETCODE rc;

    /* EXEC SQL CONNECT TO :server USER :uid USING :pwd; */
    /* Allocate an environment handle. */
    /* Allocate a connection handle. */
    /* Connect to a data source. */
    /* Allocate a statement handle. */

    SQLAllocEnv(&henv);
    SQLAllocConnect(henv, &hdbc);
    rc = SQLConnect(hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return(print_err(hdbc, SQL_NULL_HSTMT));
    SQLAllocStmt(hdbc, &hstmt);

    /* EXEC SQL CREATE TABLE NAMEID (ID integer, NAME varchar(50)); */
    /* Execute the SQL statement. */

    strcpy(create, "CREATE TABLE NAMEID (ID INTEGER, NAME
        VARCHAR(50))");
    rc = SQLExecDirect(hstmt, create, SQL_NTS);
```

```

if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
    return(print_err(hdbc, hstmt));

/* EXEC SQL COMMIT WORK; */
/* Commit the table creation. */

/* Note that the default transaction mode for drivers that support */
/* SQLSetConnectOption is auto-commit and SQLTransact has no effect */

SQLTransact(hdbc, SQL_COMMIT);

/* EXEC SQL INSERT INTO NAMEID VALUES (:id, :name); */
/* Show the use of the SQLPrepare/SQLExecute method: */
/* Prepare the insertion and bind parameters. */
/* Assign parameter values. */
/* Execute the insertion. */

lstrcpy(insert, "INSERT INTO NAMEID VALUES (?, ?)");
if (SQLPrepare(hstmt, insert, SQL_NTS) != SQL_SUCCESS)
    return(print_err(hdbc, hstmt));
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
0, 0, &id, 0, NULL);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
MAX_NAME_LEN, 0, name, 0, NULL);
id=500;
lstrcpy(name, "Babbage");
if (SQLExecute(hstmt) != SQL_SUCCESS)
    return(print_err(hdbc, hstmt));

/* EXEC SQL COMMIT WORK; */
/* Commit the insertion. */

SQLTransact(hdbc, SQL_COMMIT);

/* EXEC SQL DECLARE c1 CURSOR FOR SELECT ID, NAME FROM NAMEID; */
/* EXEC SQL OPEN c1; */
/* Show the use of the SQLExecDirect method. */
/* Execute the selection. */
/* Note that the application does not declare a cursor. */

lstrcpy(select, "SELECT ID, NAME FROM NAMEID");
if (SQLExecDirect(hstmt, select, SQL_NTS) != SQL_SUCCESS)
    return(print_err(hdbc, hstmt));

/* EXEC SQL FETCH c1 INTO :id, :name; */
/* Bind the columns of the result set with SQLBindCol. */
/* Fetch the first row. */

SQLBindCol(hstmt, 1, SQL_C_SLONG, &id, 0, NULL);
SQLBindCol(hstmt, 2, SQL_C_CHAR, name, (SDWORD)sizeof(name), &namelen);
SQLFetch(hstmt);

/* EXEC SQL COMMIT WORK; */
/* Commit the transaction. */

```

## Static SQL Example

```
SQLTransact(hdbc, SQL_COMMIT);

/* EXEC SQL CLOSE c1;          */
/* Free the statement handle. */

SQLFreeStmt(hstmt, SQL_DROP);

/* EXEC SQL DISCONNECT;      */
/* Disconnect from the data source. */
/* Free the connection handle. */
/* Free the environment handle. */

SQLDisconnect(hdbc);
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);

return(0);
}
```

## Interactive Ad-Hoc Query Example

The following example illustrates how an application can determine the nature of the result set prior to retrieving results:

```
#include "sql.h"
#include <string.h>
#include <stdlib.h>

#define MAXCOLS 100
#define max(a,b) (a>b?a:b)

int print_err(HDBC hdbc, HSTMT hstmt);
UDWORD display_size(SWORD coltype, UDWORD collen, UCHAR *colname);

example2(server, uid, pwd, sqlstr)
UCHAR * server;
UCHAR * uid;
UCHAR * pwd;
UCHAR * sqlstr;
{
    int i;
    HENV henv;
    HDBC hdbc;
    HSTMT hstmt;
    UCHAR errmsg[256];
    UCHAR colname[32];
    SWORD coltype;
    SWORD colnamelen;
    SWORD nullable;
    UDWORD collen[MAXCOLS];
    SWORD scale;
    SDWORD outlen[MAXCOLS];
    UCHAR * data[MAXCOLS];
    SWORD nresultcols;
    SDWORD rowcount;
    RETCODE rc;

    /* Allocate environment and connection handles. */
    /* Connect to the data source. */
    /* Allocate a statement handle. */
    SQLAllocEnv(&henv);
    SQLAllocConnect(henv, &hdbc);
    rc = SQLConnect(hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return(print_err(hdbc, SQL_NULL_HSTMT));
    SQLAllocStmt(hdbc, &hstmt);
    /* Execute the SQL statement. */
    if (SQLExecDirect(hstmt, sqlstr, SQL_NTS) != SQL_SUCCESS)
        return(print_err(hdbc, hstmt));

    /* See what kind of statement it was.If there are no result */
}
```

## Interactive Ad-Hoc Query Example

```
/* columns, the statement is not a SELECT statement. If the */
/* number of affected rows is greater than 0, the statement was */
/* probably an UPDATE, INSERT, or DELETE statement, so print the */
/* number of affected rows. If the number of affected rows is 0, */
/* the statement is probably a DDL statement, so print that the */
/* operation was successful and commit it. */

SQLNumResultCols(hstmt, &nresultcols);
if (nresultcols == 0) {
    SQLRowCount(hstmt, &rowcount);
    if (rowcount > 0) {
        printf("%ld rows affected.\n", rowcount);
    } else {
        printf("Operation successful.\n");
    }
    SQLTransact(hdbc, SQL_COMMIT);

    /* Otherwise, display the column names of the result set and use the */
    /* display_size() function to compute the length needed by each data */
    /* type. Next, bind the columns and specify all data will be */
    /* converted to char. Finally, fetch and print each row, printing */
    /* truncation messages as necessary. */

} else {
    for (i = 0; i < nresultcols; i++) {
        SQLDescribeCol(hstmt, i + 1, colname, (SWORD)sizeof(colname),
            &colnamelen, &coltype, &collen[i], &scale,
            &nullable);
        collen[i] = display_size(coltype, collen[i], colname);
        printf("%*.s", collen[i], collen[i], colname);
        data[i] = (UCHAR *) malloc(collen[i] + 1);
        SQLBindCol(hstmt, i + 1, SQL_C_CHAR, data[i], collen[i],
            &outlen[i]);
    }
    while (TRUE) {
        rc = SQLFetch(hstmt);
        if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO) {
            errmsg[0] = '\0';
            for (i = 0; i < nresultcols; i++)
                if (outlen[i] == SQL_NULL_DATA) {
                    strcpy(data[i], "NULL");
                } else if (outlen[i] >= collen[i]) {
                    sprintf(&errmsg[strlen(errmsg)],
                        "%d chars truncated, col %d\n",
                        *outlen[i] - collen[i] + 1,
                        colnum);
                }
            printf("%*.s ", collen[i], collen[i], data[i]);
        }
        printf("\n%s", errmsg);
    } else {
        break;
    }
}
}
```



```

/* Free the data buffers. */
for (i = 0; i < nresultcols; i++) {
    free(data[i]);
}

SQLFreeStmt(hstmt, SQL_DROP ); /* Free the statement handle.      */
SQLDisconnect(hdbc);          /* Disconnect from the data source. */
SQLFreeConnect(hdbc);        /* Free the connection handle.    */
SQLFreeEnv(henv);           /* Free the environment handle.   */

return(0);
}
/*****
/* The following function is included for completeness, but */
/* is not relevant for understanding the function of ODBC. */
*****/

#define MAX_NUM_PRECISION 15

/* Define max length of char string representation of number as:
/* = max(precision) + leading sign + E + exp sign + max exp length */
/* = 15          + 1          + 1 + 1          + 2          */
/* = 15 + 5
*/

#define MAX_NUM_STRING_SIZE (MAX_NUM_PRECISION + 5)

UDWORD display_size(coltype, collen, colname)
SWORD  coltype;
UDWORD collen;
UCHAR * colname;
{
switch (coltype) {

    case SQL_CHAR:
    case SQL_VARCHAR:
        return(max(collen, strlen(colname)));

    case SQL_SMALLINT:
        return(max(6, strlen(colname)));

    case SQL_INTEGER:
        return(max(11, strlen(colname)));

    case SQL_DECIMAL:
    case SQL_NUMERIC:
    case SQL_REAL:
    case SQL_FLOAT:
    case SQL_DOUBLE:
        return(max(MAX_NUM_STRING_SIZE, strlen(colname)));

    /* Note that this function only supports the core data types. */
    default:
        printf("Unknown datatype, %d\n", coltype);
        return(0);
}
}

```



---

# Designing Performance-Oriented Applications

Connecting to a Data Source . . . . .	11-4
Executing Calls . . . . .	11-11
SQLPrepare/SQLExecute Versus SQLExecDirect . . . . .	11-11
Committing Data . . . . .	11-16



**T**his chapter provides guidelines for designing and coding performance-oriented INFORMIX-CLI applications. These guidelines are not hard theorems but general advice that apply to ODBC applications. The guidelines address areas in which inefficient performance can be improved.

---

## Areas in Which Performance is Impacted

A number of areas can slow performance. This section aims to identify the impacted performance area and provide solutions to increase performance levels.

---

Performance Issue	Action
Network communication is slow.	Reduce network traffic as much as possible.
The process of evaluating complex SQL queries on the DBMS server might be slow and might reduce concurrency.	Simplify queries as much as possible.
Excessive calls from the application to the driver decrease performance.	Optimize the application to driver interaction.
Disk I/O is slow.	Limit disk I/O to improve performance.

---

---

## Connecting to a Data Source

Connection management is extremely important to application performance. Connecting to a data source is very expensive and should be avoided after you establish an initial connection. Designers should optimize applications by connecting once and using multiple statement handles instead of performing multiple connections.

Some ODBC applications are designed so that they call informational gathering routines that have no record of already-attached connection handles. For example, some applications establish a connection and then call a routine in a separate DLL or shared library that re-attaches and gathers up-front information about the driver to be used later in the application. Although gathering driver information at connect time is a good algorithm, it should not be minimized by connecting twice to get this information. At least one popular ODBC-enabled application connects a second time to gather driver information but *never* disconnects the second connection. Applications that are designed as separate entities should pass the already-connected *hdbc* pointer to the data collection routine instead of establishing a second connection.

Another poor algorithm is to connect and disconnect several times throughout your application to perform SQL statements. Connection handles can have multiple statement handles associated with them. Statement handles are defined to be memory storage for information about SQL statements.

Applications should use *statement* handles to manage multiple SQL statements.

Connection and statement handling should not be delayed until implementation. Spending time in the design phase on connection management makes your application perform better and makes it more maintainable.

---

## Retrieving Information about a Data Source

Catalog functions are relatively slow compared to all other ODBC functions. In addition, **SQLGetTypeInfo** is also a potentially expensive ODBC function and is included in this discussion of catalog functions. For a list of the catalog functions that INFORMIX-CLI supports, see [“Retrieving Information About the Data-Source Catalog” on page 6-9](#).

### Minimize the use of Catalog Functions

Although almost no ODBC application can be written without catalog functions, their use should be minimized. To return all result column information *mandated* by the ODBC specification, a driver might have to perform multiple queries, joins, subqueries, and/or unions in order to return the necessary result set for a single call to a catalog function. Frequent use of catalog functions in an application will likely result in poor performance.

If possible, applications should cache information returned from catalog functions so that multiple executions are not needed. For example, call **SQLGetTypeInfo** once in the application and cache the elements of the result set on which your application depends. Probably no application uses all elements of the result set generated by a catalog function, so the cache of information should not be difficult to maintain.

### Supply Non-Null Arguments to Catalog Functions

Passing null arguments to catalog functions results in time-consuming queries being generated by the driver. In addition, network traffic potentially increases due to unwanted result-set information. Always supply as many non-null arguments to catalog functions as possible.

Because catalog functions are slow, applications should invoke them as efficiently as possible. Many applications pass the least amount of non-null arguments necessary for the function to return success. Consider a call to **SQLTables** in which the application requests information about a table named **Customers**. Many times this call is coded similarly to the following example.

```
rc = SQLTables (NULL, NULL, NULL, NULL, "Customers", SQL_NTS,  
              NULL);
```

A driver could turn this **SQLTables** call into SQL similar to the following commands:

```
SELECT ... FROM SysTables WHERE TableName = 'Customers'  
        UNION ALL  
SELECT ... FROM SysViews WHERE ViewName = 'Customers' UNION  
ALL  
SELECT ... FROM SysSynonyms WHERE SynName = 'Customers'  
ORDER BY ...
```

In some circumstances, not much information is known about the object for which you are requesting information, but in many cases at least some information is known. Any additional information that the application can send the driver when calling catalog functions can result in improved performance and reliability.

Suppose in the previous example that three **Customers** tables were returned in the result set: one owned by the user, one owned by sales, and one that was a view created by management. Although it might be obvious that the intent of the application is to use the one owned by the user, it might not be obvious to the user which table to choose. If the application had specified the *OwnerName* argument for the **SQLTables** call, then reliability is improved (only one table returned) and performance increases. (Less network traffic is required to return only one result row, and unwanted rows are filtered by the DBMS.) In addition, if the *TableType* argument can be supplied, then the SQL sent to the server can be optimized from a three-query union to a singleton SELECT similar to the following command:

```
SELECT ... FROM SysTables WHERE TableName = 'Customers' and  
        Owner = 'Beth'
```



## **Avoid Using SQLColumns to Determine Table Characteristics**

Avoid using **SQLColumns** to determine characteristics about a table. Instead, use a dummy query with **SQLDescribeCol**. In both cases, the application sends a query to the server.

When the application calls **SQLColumns**, the database server must evaluate the query and form a result set that must then be sent back to the client. When the application makes a dummy query with **SQLDescribeCol**, the database server does not execute the query; it only prepares it. Thus, no results are sent back to the client.

Consider an ad-hoc application that allows the user to choose the columns that will be selected. Should the application use **SQLColumns** to return information about the columns to the user or instead prepare a dummy query and call **SQLDescribeCol**?

### **Case 1: SQLColumns Method**

```
rc = SQLColumns (... "UnknownTable" ...);  
// This call to SQLColumns will generate a query to the system  
// catalogs... possibly a join which must be prepared,  
// executed, and produce a result set  
rc = SQLBindCol (...);  
rc = SQLExtendedFetch (...);  
// user must retrieve N rows from the server  
// N = # result columns of UnknownTable  
// result column information has now been obtained
```

### **Case 2: SQLDescribeCol Method**

```
// prepare dummy query  
rc = SQLPrepare (... "SELECT * from UnknownTable  
WHERE 1 = 0" ...);  
// query is never executed on the server - only prepared  
rc = SQLNumResultCols (...);  
for (irow = 1; irow <= NumColumns; irow++) {  
    rc = SQLDescribeCol (...)  
    // + optional calls to SQLColAttributes  
}  
// result column information has now been obtained  
// Note we also know the column ordering within the table!  
// This information cannot be  
// assumed from the SQLColumns example.
```

In both cases a query is sent to the server, but in Case 1 the query must be evaluated and form a result set that must be sent to the client. Clearly, Case 2 is the better performing model.

---

## Retrieving Data

Limit the amount of data retrieved and reduce the call load to increase the performance of an application.

### Retrieving Long Data

Retrieving long data (SQL\_LONGVARCHAR and SQL\_LONGVARBINARY data) across the network is resource intensive and thus slow. Applications should avoid requesting long data unless it is absolutely necessary. If the user does wish to see these result items, then the application can requery the database specifying only the long columns in the select list.

How often do users want to see long data? By default, it is generally acceptable not to retrieve long data or binary data because most users do not want to see such information. If the user does wish to see these result items, then the application can requery the database specifying only the long columns in the select list. This method allows the average user to retrieve the result set without having to pay a high performance penalty for intense network traffic.

Although the optimal method is to exclude long data from the select list, some applications do not formulate the select list before they send the query to the ODBC driver (that is, some applications simply `SELECT * FROM <table name> . . .`). If the select list contains long data then some drivers *must* retrieve that data at fetch time even if the application does not bind the long data in the result set. If possible, the designer should attempt to implement a method that does not retrieve all columns of the table.

## Use `SQLSetStmtOption` to Reduce the Size of Data Retrieved

To reduce the size of any data being retrieved to some manageable limit, call `SQLSetStmtOption` with the `SQL_MAX_SIZE` option. This option reduces network traffic and improves performance by allowing the driver to communicate to the DBMS backend server that only *Z* bytes of data are pertinent to the client.

While eliminating `SQL_LONGVARCHAR` and `SQL_LONGVARBINARY` data from the result set is ideal in terms of performance, many times long data must be retrieved. Consider, however, that most users do not want to see 100 kilobytes or more of textual information on the screen. What techniques, if any, are available to limit the amount of data retrieved?

Many application developers mistakenly assume that if they call `SQLGetData` with a container of size *x* that the ODBC driver only retrieves *x* bytes of information from the server. Because `SQLGetData` can be called multiple times for any one column, most drivers optimize their network use by retrieving long data in large chunks and then returning it to the user when requested.

**Example:**

```
char CaseContainer[1000];
...
rc = SQLExecDirect (hstmt, "SELECT CaseHistory FROM Cases
WHERE
    CaseNo = 71164", SQL_NTS);
...
rc = SQLFetch (hstmt);
rc = SQLGetData (hstmt, 1, CaseContainer, (SQLWORD)
sizeof(CaseContainer), ...);
```

At this point, it is more likely that an ODBC driver retrieves 64 kilobytes of information from the server instead of 1000 bytes. One 64 kilobytes retrieval is less expensive than sixty-four 1000 byte retrievals in terms of network access. Unfortunately, the application might not call `SQLGetData` again; thus, the first and only retrieval of `CaseHistory` was slowed by the fact that 64 kilobytes of data had to be sent across the network.

Many ODBC drivers allow limiting the amount of data retrieved across the network by supporting the statement option `SQL_MAX_SIZE`. This option allows the driver to communicate to the database server that only *Z* bytes of data are pertinent to the client. The server responds by sending only the first *Z* bytes of data for *all* result columns. This optimization greatly reduces network traffic and thus improves performance of the client. The preceding example returned just one row, but consider the case where 100 rows are returned in the result set. The performance improvement is substantial.

## Use `SQLBindCol` to Reduce the Call Load

To improve performance, retrieve data through bound columns (`SQLBindCol`) instead of using `SQLGetData` to reduce the ODBC call load.

Consider the following pseudo-code fragment:

```
rc = SQLExecDirect (hstmt, "SELECT <20 columns> FROM Employees
    WHERE HireDate >= ?", SQL_NTS);
do {
    rc = SQLFetch (hstmt);
    // call SQLGetData 20 times
} while ((rc == SQL_SUCCESS) || (rc == SQL_SUCCESS_WITH_INFO));
```

Suppose the query returns 90 result rows. The number of ODBC calls made is > 1890 (20 calls to `SQLGetData` x 90 result rows + 91 calls to `SQLFetch`).

Consider the same scenario that uses `SQLBindCol` instead of `SQLGetData`.

```
rc = SQLExecDirect (hstmt, "SELECT <20 columns> FROM Employees
    WHERE HireDate >= ?", SQL_NTS);
// call SQLBindCol 20 times
do {
    rc = SQLFetch (hstmt);
} while ((rc == SQL_SUCCESS) || (rc == SQL_SUCCESS_WITH_INFO));
```

The number of ODBC calls made is reduced from greater than 1890 to about 110 (20 calls to `SQLBindCol` + 91 calls to `SQLFetch`). In addition to reducing the call load, many drivers optimize use of `SQLBindCol` by binding result information directly from the DBMS into the user's buffer. That is, instead of the driver retrieving information into a container and then copying that information to the user's buffer, the driver simply requests that the information from the server be placed directly into the user's buffer.

## Use **SQLExtendedFetch** instead of **SQLFetch**

Use **SQLExtendedFetch** instead of **SQLFetch** to retrieve data. The ODBC call load decreases (resulting in better performance), and the code is less complex (resulting in more maintainable code).

Again consider the preceding example using **SQLExtendedFetch** instead of **SQLFetch**:

```
rc = SQLSetStmtOption (hstmt, SQL_ROWSET_SIZE, 100);
// use arrays of 100 elements
rc = SQLExecDirect (hstmt, "SELECT <20 columns> FROM
    Employees WHERE HireDate >= ?", SQL_NTS);
// call SQLBindCol 1 time specifying row-wise binding
do {
rc = SQLExtendedFetch (hstmt, SQL_FETCH_NEXT, 0, &RowsFetched,
    RowStatus);
} while ((rc == SQL_SUCCESS) || (rc == SQL_SUCCESS_WITH_INFO));
```

The number of ODBC calls made by the application is reduced from 110 in the previous example to 4 calls (**1 SQLSetStmtOption + 1 SQLExecDirect + 1 SQLBindCol + 1 SQLExtendedFetch**). The total savings changes from an initial call load of more than 1890 ODBC calls in the first presentation of the example to 4 calls. In addition to reducing the call load, many ODBC drivers retrieve data from the server in arrays that further improve the performance by reducing network traffic.

---

## Executing Calls

ODBC drivers are optimized based on the perceived use of the functions that are being executed. This section discusses differences that can affect the performance of your application.

### SQLPrepare/SQLExecute Versus SQLExecDirect

Use **SQLExecDirect** for queries that will be executed once and **SQLPrepare/SQLExecute** for queries that will be executed more than once.

**SQLPrepare/SQLExecute** is optimized for multiple executions of a statement that most likely uses parameter markers. **SQLExecDirect** is optimized for a single execution of an SQL statement. Unfortunately, more than 75 percent of all ODBC applications use **SQLPrepare/SQLExecute** *exclusively*.

The pitfall of always coding **SQLPrepare/SQLExecute** can be understood better by considering an ODBC driver that implements **SQLPrepare** by creating a stored procedure on the server that contains the prepared statement. Creating a stored procedure has substantial overhead, but the ODBC driver assumes that the statement will be *executed* multiple times. Although stored procedure creation is relatively expensive, execution is minimal because the query is parsed and optimization paths are stored when the procedure is created. Using **SQLPrepare/SQLExecute** for a statement that will be executed only once with such an ODBC driver will result in unneeded overhead. Furthermore, applications that use **SQLPrepare/SQLExecute** for large, single-execution query batches will almost certainly exhibit poor performance when they are used with ODBC drivers.

Similar arguments can be used to show applications that always use **SQLExecDirect** cannot perform as well as those that logically use a combination of **SQLPrepare/SQLExecute** and **SQLExecDirect** sequences.

## Using SQLPrepare and Multiple SQLExecute Calls

Applications that use **SQLPrepare** and multiple **SQLExecute** calls should use **SQLParamOptions**. Passing arrays of parameter values reduces the ODBC call load and greatly reduces network traffic.

Consider the following pseudocode for inserting data in bulk:

```
rc = SQLPrepare (hstmt, "INSERT INTO DailyLedger (...) VALUES
    (?,?,...)", SQL_NTS);
// bind parameters
...
do {
    // read ledger values into bound parameter buffers
    ...
    rc = SQLExecute (hstmt);      // insert row
} while ! (eof);
```

If there are 100 rows to insert, **SQLExecute** is called 100 times, resulting in 100 network requests to the server. Consider, however, an algorithm that uses parameter arrays by calling **SQLParamOptions**.

```
rc = SQLPrepare (hstmt, "INSERT INTO DailyLedger (...) VALUES
    (?,?,...)", SQL_NTS);
rc = SQLParamOptions (hstmt, (UDWORD) 50, &CurrentRow);
// pass 50 parameters per execute
// bind parameters
...
do {
    // read up to 50 ledger values into bound parameter buffers
    ...
    rc = SQLExecute (hstmt);    // insert row
```

The call load has been reduced from 100 executions to just two **SQLExecute** calls; furthermore, network traffic is reduced considerably. To achieve high performance, applications should contain algorithms for using **SQLParamOptions**. **SQLParamOptions** is ideal for copying data into new tables or loading tables in bulk.

---

## Updating Data

Use positioned updates and deletes and **SQLSpecialColumns** to increase the performance of your application.

### Use Positioned Updates and Deletes

Designing an efficient method for updating data is challenging. Although positioned updates do not apply to all types of applications, attempt to use positioned updates and deletes whenever possible. Positioned updates (via **UPDATE WHERE CURRENT OF** cursor) allow you to update data simply by positioning the database cursor to the appropriate row to be changed and signal the driver to “change the data here.” You are not forced to build a complex SQL statement but simply required to supply the data that is to be changed.

Besides making the code more maintainable, positioned updates typically result in improved performance. Because the database server is already positioned on the row (for the `SELECT` statement currently in process), expensive operations to locate the row to be changed are not needed. If the row must be located, the server typically has an internal pointer to the row available (for example, `ROWID`).

## Use `SQLSpecialColumns` to Determine the Optimal Set of Columns

Use `SQLSpecialColumns` to determine the optimal set of columns to use in the `WHERE` clause for updating data. Many times pseudo-columns provide the fastest access to the data, and these columns can only be determined by using `SQLSpecialColumns`.

Many applications cannot be designed to take advantage of positioned updates and deletes. These applications typically update data by forming a `WHERE` clause that consists of some subset of the column values returned in the result set. Some applications might formulate the `WHERE` clause by using all searchable result columns or by calling `SQLStatistics` to find columns that might be part of a unique index. These methods typically work but can result in fairly complex queries.

Consider the following:

```
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,  
    address, city, state, zip FROM emp", SQL_NTS);  
// fetchdata  
...  
rc = SQLExecDirect (hstmt, "UPDATE EMP SET ADDRESS = ?  
    WHERE first_name = ? and last_name = ? and ssn = ?  
    and address = ? and city = ? and state = ? and zip = ?",  
    SQL_NTS);  
// fairly complex query
```



Applications should call **SQLSpecialColumns**/SQL\_BEST\_ROWID to retrieve the optimal set of columns (possibly a pseudo-column) that identifies any given record. Many databases support special columns that are not explicitly defined by the user in the table definition but are “hidden” columns of every table (for example, ROWID, TID, and so on). These pseudo-columns almost always provide the fastest access to the data because they typically are pointers to the exact location of the record. Because pseudo-columns are not part of the explicit table definition, they are not returned from **SQLColumns**. The only method of determining if pseudo-columns exist is to call **SQLSpecialColumns**.

Consider the previous example again:

```
...
rc = SQLSpecialColumns (hstmt, ..... 'emp', ...);
...
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
    address, city, state, zip, ROWID FROM emp", SQL_NTS);
// fetch data and probably "hide" ROWID from the user
...
rc = SQLExecDirect (hstmt, "UPDATE emp SET address = ? WHERE
    ROWID = ?", SQL_NTS);
// fastest access to the data!
```

If your data source does not contain special pseudo-columns, the result set of **SQLSpecialColumns** consists of the columns of the optimal unique index on the specified table (if a unique index exists); therefore, your application need not additionally call **SQLStatistics** to find the smallest unique index.

---

## Committing Data

Committing data is extremely disk I/O intensive and thus slow. Always turn auto-commit off if the driver can support transactions.

What is actually involved in a commit? At commit time, the DBMS server must flush back to disk every data page that contains updated or new data. This is not a sequential write but a searched write to replace existing data already in the table. By default, auto-commit is on when connecting to a data source. Auto-commit mode is typically detrimental to performance because of the amount of disk I/O needed to commit *every* operation.

Further reducing performance, some DBMS servers do not provide an auto-commit mode. For this type of server, the ODBC driver must explicitly issue a COMMIT statement and perhaps a BEGIN TRANSACTION for *every* operation sent to the server. In addition to the large amount of disk I/O required to support auto-commit mode, we must also pay a performance penalty for up to three network requests for every statement issued by an application in this scenario.

---

# Function Summary

INFORMIX-CLI Function Summary . . . . .	12-3
Connecting to a Data Source . . . . .	12-4
Obtaining Information About a Driver and Data Source. . . . .	12-4
Setting and Retrieving Driver Options. . . . .	12-5
Preparing SQL Requests. . . . .	12-5
Submitting SQL Requests . . . . .	12-6
Retrieving Results and Information about Results . . . . .	12-7
Obtaining Information About Data-Source System Tables . . . . .	12-8
Terminating a Statement . . . . .	12-9
Terminating a Connection . . . . .	12-9
Setup Shared-Library Function Summary . . . . .	12-10
Translation Shared-Library Function Summary . . . . .	12-10



# T

his chapter summarizes the functions that INFORMIX-CLI-enabled applications and related software use:

- INFORMIX-CLI functions
- Setup shared-library functions
- Translation shared-library functions

---

## INFORMIX-CLI Function Summary

The following tables list INFORMIX-CLI functions according to the type of task that the functions perform. The tables include function name, conformance designation, and a brief description of each function. For more information about conformance designations, see [“API Conformance Level of INFORMIX-CLI” on page 1-12](#). For more information about the syntax and semantics for each function, see [Chapter 13, “INFORMIX-CLI Function Reference.”](#)

An application can call the **SQLGetInfo** function to obtain conformance information about a driver. To obtain information about support for a specific function in a driver, an application can call **SQLGetFunctions**.

## Connecting to a Data Source

The following table describes functions that connect to a data source.

Function Name	ODBC Conformance	Purpose
<b>SQLAllocEnv</b>	Core	Obtains an environment handle. One environment handle is used for one or more connections.
<b>SQLAllocConnect</b>	Core	Obtains a connection handle.
<b>SQLConnect</b>	Core	Connects to a specific driver by data source name, user ID, and password.
<b>SQLDriverConnect</b>	Level 1	Connects to a specific driver by connection string or requests that the driver manager and driver display connection dialog boxes for the user.
<b>SQLBrowseConnect</b>	Level 2	Returns successive levels of connection attributes and valid attribute values. When a value has been specified for each connection attribute, connects to the data source.

## Obtaining Information About a Driver and Data Source

The following table describes functions that obtain information about a driver and data source.

Function Name	ODBC Conformance	Purpose
<b>SQLDataSources</b>	Level 2	Returns the list of available data sources.
<b>SQLDrivers</b>	Level 2	Returns the list of installed drivers and their attributes.

(1 of 2)

Function Name	ODBC Conformance	Purpose
<b>SQLGetInfo</b>	Level 1	Returns information about a specific driver and data source.
<b>SQLGetFunctions</b>	Level 1	Returns supported driver functions.
<b>SQLGetTypeInfo</b>	Level 1	Returns information about supported data types.

(2 of 2)

## Setting and Retrieving Driver Options

The following table describes functions that set and retrieve driver options.

Function Name	ODBC Conformance	Purpose
<b>SQLSetConnectOption</b>	Level 1	Sets a connection option.
<b>SQLGetConnectOption</b>	Level 1	Returns the value of a connection option.
<b>SQLSetStmtOption</b>	Level 1	Sets a statement option.
<b>SQLGetStmtOption</b>	Level 1	Returns the value of a statement option.

## Preparing SQL Requests

The following table describes functions that prepare SQL requests.

Function Name	ODBC Conformance	Purpose
<b>SQLAllocStmt</b>	Core	Allocates a statement handle.
<b>SQLPrepare</b>	Core	Prepares an SQL statement for later execution.

(1 of 2)

Function Name	ODBC Conformance	Purpose
<b>SQLBindParameter</b>	Level 1	Assigns storage for a parameter in an SQL statement.
<b>SQLParamOptions</b>	Level 2	Specifies the use of multiple values for parameters.
<b>SQLGetCursorName</b>	Core	Returns the cursor name associated with a statement handle.
<b>SQLSetCursorName</b>	Core	Specifies a cursor name.
<b>SQLSetScrollOptions</b>	Level 2	Sets options that control cursor behavior.

(2 of 2)

## Submitting SQL Requests

The following table describes functions that submit SQL requests.

Function Name	ODBC Conformance	Purpose
<b>SQLExecute</b>	Core	Executes a prepared statement.
<b>SQLExecDirect</b>	Core	Executes a statement.
<b>SQLNativeSql</b>	Level 2	Returns the text of an SQL statement as translated by the driver.
<b>SQLNumParams</b>	Level 2	Returns the number of parameters in a statement.
<b>SQLParamData</b>	Level 1	Used with <b>SQLPutData</b> to supply parameter data at execution time. (Useful for long data values.)
<b>SQLPutData</b>	Level 1	Sends part or all of a data value for a parameter. (Useful for long data values.)



## Retrieving Results and Information about Results

The following table describes functions that retrieve results and information about results.

Function Name	ODBC Conformance	Purpose
<b>SQLRowCount</b>	Core	Returns the number of rows affected by an insert, update, or delete request.
<b>SQLNumResultCols</b>	Core	Returns the number of columns in the result set.
<b>SQLDescribeCol</b>	Core	Describes a column in the result set.
<b>SQLColAttributes</b>	Core	Describes attributes of a column in the result set.
<b>SQLBindCol</b>	Core	Assigns storage for a result column and specifies the data type.
<b>SQLFetch</b>	Core	Returns a result row.
<b>SQLExtendedFetch</b>	Level 2	Returns multiple result rows.
<b>SQLGetData</b>	Level 1	Returns part or all of one column of one row of a result set. (Useful for long data values.)
<b>SQLMoreResultsr</b>	Level 2	Determines whether more result sets are available and, if so, initializes processing for the next result set.
<b>SQLError</b>	Core	Returns additional error or status information.

## Obtaining Information About Data-Source System Tables

The following table describes functions that obtain information about data-source system tables (catalog functions).

Function Names	ODBC Conformance	Purpose
<b>SQLColumnPrivileges</b>	Level 2	Returns a list of columns and associated privileges for one or more tables.
<b>SQLColumns</b>	Level 1	Returns the list of column names in specified tables.
<b>SQLPrimaryKeys</b>	Level 2	Returns the list of column name(s) that comprise the primary key for a table.
<b>SQLProcedures</b>	Level 2	Returns a list of procedure names stored in a specific data source.
<b>SQLSpecialColumns</b>	Level 1	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
<b>SQLStatistics</b>	Level 1	Returns statistics about a single table and the list of indexes associated with the table.
<b>SQLTablePrivileges</b>	Level 2	Returns a list of tables and the privileges associated with each table.
<b>SQLTables</b>	Level 1	Returns the list of table names stored in a specific data source.

## Terminating a Statement

The following table describes functions that terminate a statement.

Function Name	ODBC Conformance	Purpose
<b>SQLFreeStmt</b>	Core	Ends statement processing and closes the associated cursor, discards pending results, and, optionally, frees all resources associated with the statement handle.
<b>SQLCancel</b>	Core	Cancels an SQL statement.
<b>SQLTransact</b>	Core	Commits or rolls back a transaction.

## Terminating a Connection

The following table describes functions that terminate a connection.

Function Name	ODBC Conformance	Purpose
<b>SQLDisconnect</b>	Core	Closes the connection.
<b>SQLFreeConnect</b>	Core	Releases the connection handle.
<b>SQLFreeEnv</b>	Core	Releases the environment handle.

INFORMIX-CLI does not support the following ODBC functions:

- **SQLDescribeParam**
- **SQLForeignKeys**
- **SQLProcedurColumns**
- **SQLSetPos**

---

## Setup Shared-Library Function Summary

The following table describes setup shared-library functions. For more information about the syntax and semantics for each function, see [Chapter 14, “Setup Shared Library Function Reference.”](#)

---

Task	Function Name	Purpose
Setting up data sources and translators	<b>ConfigDSN</b>	Adds, modifies, or deletes a data source.
	<b>ConfigTranslator</b>	Returns a default translation option.

---

---

## Translation Shared-Library Function Summary

The following table describes translation shared-library functions. For more information about the syntax and semantics for each function, see [Chapter 15, “Translation Shared Library Function Reference.”](#)

---

Task	Function name	Purpose
Translating data	<b>SQLDriverToDataSource</b>	Translates all data that flows from the driver to the data source.
	<b>SQLDataSourceToDriver</b>	Translates all data that flows from the data source to the driver.

---

---

# INFORMIX-CLI Function Reference

Arguments . . . . .	13-6
INFORMIX-CLI Include Files . . . . .	13-9
Diagnostics . . . . .	13-9
Tables and Views . . . . .	13-9
Catalog Functions . . . . .	13-10
Search Pattern Arguments . . . . .	13-10
SQLAllocConnect . . . . .	13-11
SQLAllocEnv . . . . .	13-14
SQLAllocStmt . . . . .	13-16
SQLBindCol . . . . .	13-18
SQLBindParameter . . . . .	13-25
SQLBrowseConnect . . . . .	13-40
SQLCancel . . . . .	13-51
SQLColAttributes . . . . .	13-54
SQLColumnPrivileges . . . . .	13-62
SQLColumns . . . . .	13-68
SQLConnect . . . . .	13-77
SQLDataSources . . . . .	13-83
SQLDescribeCol . . . . .	13-87
Related Functions . . . . .	13-92
SQLDisconnect . . . . .	13-92
SQLDriverConnect . . . . .	13-95
SQLDrivers . . . . .	13-106
SQLError . . . . .	13-110
SQLExecDirect . . . . .	13-113
SQLExecute . . . . .	13-122

SQLExtendedFetch . . . . .	13-129
SQLFetch . . . . .	13-146
SQLFreeConnect . . . . .	13-152
SQLFreeEnv . . . . .	13-154
SQLFreeStmt. . . . .	13-156
SQLGetConnectOption . . . . .	13-160
SQLGetCursorName . . . . .	13-163
SQLGetData . . . . .	13-166
SQLGetFunctions . . . . .	13-176
SQLGetInfo . . . . .	13-183
SQLGetStmtOption . . . . .	13-215
SQLGetTypeInfo . . . . .	13-218
SQLMoreResults . . . . .	13-227
SQLNativeSql . . . . .	13-230
SQLNumParams . . . . .	13-233
SQLNumResultCols . . . . .	13-235
SQLParamData . . . . .	13-238
SQLParamOptions . . . . .	13-241
SQLPrepare . . . . .	13-246
SQLPrimaryKeys . . . . .	13-253
SQLProcedures . . . . .	13-257
SQLPutData . . . . .	13-265
SQLRowCount . . . . .	13-272
SQLSetConnectOption . . . . .	13-274
SQLSetCursorName . . . . .	13-283
SQLSetParam . . . . .	13-287
SQLSetScrollOptions . . . . .	13-287
SQLSetStmtOption. . . . .	13-288
SQLSpecialColumns . . . . .	13-297
SQLStatistics. . . . .	13-307
SQLTablePrivileges . . . . .	13-315
SQLTables. . . . .	13-321
SQLTransact . . . . .	13-327

# T

his chapter describes each INFORMIX-CLI function. The functions are listed alphabetically. Each function is defined as a programming-language function. The function descriptions include the following aspects:

- Conformance level
- Purpose
- Syntax
- Return codes
- Diagnostics
- Usage
- Code example (optional)
- Related functions

Error handling is described under the **SQLERROR** function description. The text associated with SQLSTATE values is included to provide a description of the condition, but it is not intended to prescribe specific text.

---

## Arguments

All INFORMIX-CLI function arguments use a naming convention of the following form:

```
[[prefix]....]tag[qualifier][suffix]
```

Optional elements are enclosed in brackets ( [ ] ) and use the following prefixes.

Prefix	Description
c	Count of
h	Handle of
i	Index of
p	Pointer to
rg	Range (array) of

The following tags are used.

Tag	Description
b	Byte
col	Column (of a result set)
dbc	Database connection
env	Environment
f	Flag (enumerated type)
par	Parameter (of an SQL statement)
row	Row (of a result set)
stmt	Statement
sz	Character string (array of characters, terminated by zero)
v	Value of unspecified type

---



Prefixes and tags combine to correspond roughly to the ODBC C types listed below. Flags (f) and byte counts (cb) do not distinguish among SWORD, UWORD, SDWORD, and UDWORD.

Combined	Prefix	Tag	ODBC C Type(s)	Description
cb	c	b	SWORD, SDWORD, UDWORD	Count of bytes
crow	c	row	SDWORD, UDWORD, UWORD	Count of rows
f	-	f	SWORD, UWORD	Flag
hdbc	h	dbc	HDBC	Connection handle
henv	h	env	HENV	Environment handle
hstmt	h	stmt	HSTMT	Statement handle
hwnd	h	wnd	HWND	Widget
ib	i	b	SWORD	Byte index
icol	i	col	UWORD	Column index
ipar	i	par	UWORD	Parameter index
irow	i	row	SDWORD, UWORD	Row index
pcb	pc	b	SWORD FAR *, SDWORD FAR *, UDWORD FAR *	Pointer to byte count
pccol	pc	col	SWORD FAR *	Pointer to column count
pcpar	pc	par	SWORD FAR *	Pointer to parameter count
pcrow	pc	row	SDWORD FAR *, UDWORD FAR *	Pointer to row count
pf	p	f	SWORD, SDWORD, UWORD	Pointer to flag
phdbc	ph	dbc	HDBC FAR *	Pointer to connection handle
phenv	ph	env	HENV FAR *	Pointer to environment handle

(1 of 2)

Combined	Prefix	Tag	ODBC C Type(s)	Description
phstmt	ph	stmt	HSTMT FAR *	Pointer to statement handle
pib	pi	b	SWORD FAR *	Pointer to byte index
pirow	pi	row	UDWORD FAR *	Pointer to row index
prgb	prg	b	PTR FAR *	Pointer to range (array) of bytes
pv	p	v	PTR	Pointer to value of unspecified type
rgb	rg	b	PTR	Range (array) of bytes
rgf	rg	f	UWORD FAR *	Range (array) of flags
sz	-	sz	UCHAR FAR *	String, zero terminated
v	-	v	UDWORD	Value of unspecified type

(2 of 2)

Qualifiers are used to distinguish specific variables of the same type. Qualifiers consist of the concatenation of one or more capitalized English words or abbreviations.

INFORMIX-CLI defines one value for the suffix *Max*, which denotes that the variable represents the largest value of its type for a given situation.

For example, the argument *cbErrorMsgMax* contains the largest possible byte count for an error message; in this case, the argument corresponds to the size in bytes of the argument *szErrorMsg*, a character string buffer. The argument *pcbErrorMsg* is a pointer to the count of bytes available to return in the argument *szErrorMsg*, not including the null termination character.



**Important:** Because characters are signed in many UNIX implementations and string arguments in INFORMIX-CLI functions are unsigned, applications that pass string objects to INFORMIX-CLI functions without casting them receive compiler warnings.

---

## INFORMIX-CLI Include Files

The files **sql.h** and **sqlext.h** contain function prototypes for all the INFORMIX-CLI functions. They also contain all type definitions and **#define** names.

---

## Diagnostics

The diagnostics provided with each function list SQLSTATE values that might be returned for the function by the driver manager or a driver. The driver can, however, return additional SQLSTATE values that arise from implementation-specific situations.

The character string value returned for an SQLSTATE consists of a two-character class value followed by a three-character subclass value. A class value of “01” indicates a warning and is accompanied by a return code of SQL\_SUCCESS\_WITH\_INFO. Class values other than “01,” except for the class “IM,” indicate an error and are accompanied by a return code of SQL\_ERROR. The class “IM” signifies warnings and errors that derive from the implementation of INFORMIX-CLI. The subclass value “000” in any class is for implementation-defined conditions within the given class. The assignment of class and subclass values is defined by ANSI SQL-92.

---

## Tables and Views

In INFORMIX-CLI functions, tables and views are interchangeable. The term *table* is used for tables and views, except where view is used explicitly.

---

## Catalog Functions

INFORMIX-CLI supports a set of functions that return information about the system tables or catalog of the data source. These functions are sometimes collectively called the *catalog functions*. For more information about catalog functions, see [“Retrieving Information About the Data-Source Catalog” on page 6-9](#). The following list shows the catalog functions:

- **SQLColumnPrivileges**
- **SQLColumns**
- **SQLPrimaryKeys**
- **SQLProcedures**
- **SQLSpecialColumns**
- **SQLStatistics**
- **SQLTablePrivileges**
- **SQLTables**

---

## Search Pattern Arguments

Each catalog function returns information in the form of a result set. The information a function returns can be constrained by a search pattern passed as an argument to that function. These search patterns can contain the underscore (`_`), the percent symbol (`%`), and a driver-defined escape character, as follows:

- The underscore represents any single character.
- The percent symbol represents any sequence of zero or more characters.
- The escape character, backslash (`\`), permits the underscore and percent metacharacters to be used as literal characters in search patterns. To use a metacharacter as a literal character in the search pattern, precede it with the escape character. To use the escape character as a literal character in the search pattern, include it twice.
- All other characters represent themselves.

For example, if the search pattern for a table name is “%A%,” the function returns all tables with names that contain the character “A.” If the search pattern for a table name is “B\_\_” (“B” followed by two underscores), the function returns all tables with names that are three characters long and start with the character “B.” If the search pattern for a table name is “%”, the function returns all tables.

If the search pattern for a table name is “ABC\%,” the function returns the table named “ABC%.” If the search pattern for a table name is “\\%,” the function returns all tables with names that start with a backslash. Failing to precede a metacharacter used as a literal with the escape character might return more results than expected. For example, if a table identifier, **MY\_TABLE**, returns as the result of a call to **SQLTables**, and an application wants to retrieve a list of columns for **MY\_TABLE** using **SQLColumns**, **SQLColumns** returns all the tables that match **MY\_TABLE**, such as **MY\_TABLE**, **MY1TABLE**, **MY2TABLE**, and so on, unless the escape character precedes the underscore.

***Important:** A zero-length search pattern matches the empty string. A search-pattern argument that is a null pointer means the search is not constrained for the argument. A null pointer and a search string of “%” should return the same values.*




---

## SQLAllocConnect

**SQLAllocConnect** allocates memory for a connection handle within the environment identified by *henv*.

### Syntax

```
RETCODE SQLAllocConnect(henv, phdbc)
```

The **SQLAllocConnect** function accepts the following arguments.

Type	Argument	Use	Description
HENV	<i>henv</i>	Input	Environment handle
HDBC FAR *	<i>phdbc</i>	Output	Pointer to storage for the connection handle

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

If **SQLAllocConnect** returns SQL\_ERROR, it sets the *hdbc* referenced by *phdbc* to SQL\_NULL\_HDBC. To obtain additional information, the application can call **SQLError** with the specified *henv* and with *hdbc* and *hstmt* set to SQL\_NULL\_HDBC and SQL\_NULL\_HSTMT, respectively.

## Diagnostics

When **SQLAllocConnect** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLAllocConnect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	(DM) The driver manager could not allocate memory for the connection handle.  The driver could not allocate memory for the connection handle.
S1009	Invalid argument value	(DM) The argument <i>phdbc</i> was a null pointer.

## Usage

A connection handle references information such as the valid statement handles on the connection and whether a transaction is currently open. To request a connection handle, an application passes the address of an *hdbc* to **SQLAllocConnect**. The driver allocates memory for the connection information and stores the value of the associated handle in the *hdbc*. On operating systems that support multiple threads, applications can use the same *hdbc* on different threads. The application passes the *hdbc* value in all subsequent calls that require an *hdbc*.

The driver manager processes the **SQLAllocConnect** function and calls the driver **SQLAllocConnect** function when the application calls **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect**. (For more information, see [“Usage” on page 13-78.](#))

If the application calls **SQLAllocConnect** with a pointer to a valid *hdbc*, the driver overwrites the *hdbc* without regard to its previous contents.

## Code Example

See **SQLBrowseConnect** and **SQLConnect**.

## Related Functions

For information about	See
Connecting to a data source	<b>SQLConnect</b>
Freeing a connection handle	<b>SQLFreeConnect</b>

## SQLAllocEnv

**SQLAllocEnv** allocates memory for an environment handle and initializes the INFORMIX-CLI call level interface for application use. An application must call **SQLAllocEnv** before it calls any other INFORMIX-CLI function.

### Syntax

```
RETCODE SQLAllocEnv(phenv)
```

The **SQLAllocEnv** function accepts the following argument.

Type	Argument	Use	Description
HENV FAR *	<i>phenv</i>	Output	Pointer to storage for the environment handle

### Return Codes

SQL\_SUCCESS or SQL\_ERROR

If **SQLAllocEnv** returns SQL\_ERROR, it sets the *henv* that *phenv* references to SQL\_NULL\_HENV. In this case, the application can assume that the error was a memory-allocation error.

### Diagnostics

A driver cannot return SQLSTATE values directly after the call to **SQLAllocEnv** because no valid handle exists with which to call **SQLError**.

Two levels of **SQLAllocEnv** functions exist, one within the driver manager and one within the driver. The driver manager does not call the driver-level function until the application calls **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect**. If an error occurs in the driver-level **SQLAllocEnv** function, the driver manager-level **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect** function returns SQL\_ERROR. A subsequent call



to **SQLERROR** with *henv*, `SQL_NULL_HDBC`, and `SQL_NULL_HSTMT` returns `SQLSTATE IM004` (the driver **SQLAllocEnv** function failed), followed by one of the following errors from the driver:

- `SQLSTATE S1000` (General error)
- A INFORMIX-CLI `SQLSTATE` value, that ranges from `S1000` to `S19ZZ`. For example, `SQLSTATE S1001` (Memory-allocation failure) indicates that the call from the driver manager to the driver-level **SQLAllocEnv** returned `SQL_ERROR`, and the *henv* from the driver manager was set to `SQL_NULL_HENV`.

For additional information about the flow of function calls between the driver manager and a driver, see [“Usage” on page 13-78](#).

## Usage

An environment handle references global information such as valid connection handles and active connection handles. To request an environment handle, an application passes the address of an *henv* to **SQLAllocEnv**. The driver allocates memory for the environment information and stores the value of the associated handle in the *henv*. On operating systems that support multiple threads, applications can use the same *hdbc* on different threads. The application passes the *henv* value in all subsequent calls that require an *henv*.

More than one *henv* should never be allocated at one time, and the application should not call **SQLAllocEnv** when a current valid *henv* exists. If the application calls **SQLAllocEnv** with a pointer to a valid *henv*, the driver overwrites the *henv* without regard to its previous contents.

When the driver manager processes the **SQLAllocEnv** function, it checks the **Trace** keyword in the ODBC options section of the `odbc.ini` file. If the keyword is set to 1, the driver manager enables tracing for all applications.

## Code Example

See **SQLBrowseConnect** and **SQLConnect**.

## Related Functions

For information about	See
Allocating a connection handle	<b>SQLAllocConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Freeing an environment handle	<b>SQLFreeEnv</b>

## SQLAllocStmt

**SQLAllocStmt** allocates memory for a statement handle and associates the statement handle with the connection that *hdbc* specifies.

An application must call **SQLAllocStmt** before it submits SQL statements.

## Syntax

```
RETCODE SQLAllocStmt(hdbc, phstmt)
```

The **SQLAllocStmt** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle
HSTMT FAR *	<i>phstmt</i>	Output	Pointer to storage for the statement handle

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_INVALID\_HANDLE, or SQL\_ERROR

If **SQLAllocStmt** returns SQL\_ERROR, it sets the *hstmt* that *phstmt* references to SQL\_NULL\_HSTMT. The application can then obtain additional information by calling **SQLError** with the *hdbc* and SQL\_NULL\_HSTMT.

## Diagnostics

When **SQLAllocStmt** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLAllocStmt** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection not open	(DM) The connection specified by the <i>hdbc</i> argument was not open. The connection process must be completed successfully (and the connection must be open) for the driver to allocate an <i>hstmt</i> .
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	(DM) The driver manager could not allocate memory for the statement handle.  The driver could not allocate memory for the statement handle.
S1009	Invalid argument value	(DM) The argument <i>phstmt</i> was a null pointer.

## Usage

A statement handle references statement information, such as network information, `SQLSTATE` values and error messages, cursor name, several result-set columns, and status information for SQL statement processing.

To request a statement handle, an application connects to a data source and then passes the address of an *hstmt* to **SQLAllocStmt**. The driver allocates memory for the statement information and stores the value of the associated handle in the *hstmt*. On operating systems that support multiple threads, applications can use the same *hdbc* on different threads. The application passes the *hstmt* value in all subsequent calls that require an *hstmt*.

If the application calls **SQLAllocStmt** with a pointer to a valid *hstmt*, the driver overwrites the *hstmt* without regard to its previous contents.

## Code Example

See **SQLBrowseConnect**, **SQLConnect**, and **SQLSetCursorName**.

## Related Functions

For information about	See
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Freeing a statement handle	<b>SQLFreeStmt</b>
Preparing a statement for execution	<b>SQLPrepare</b>

## SQLBindCol

**SQLBindCol** assigns the storage and data type for a column in a result set, as follows:

- A storage buffer that receives the contents of a column of data
- The length of the storage buffer
- A storage location that receives the actual length of the column of data returned by the fetch operation
- Data type conversion

## Syntax

```
RETCODE SQLBindCol(hstmt, icol, fCType, rgbValue, cbValueMax,  
pcbValue)
```

The **SQLBindCol** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>icol</i>	Input	Column number of result data, ordered sequentially left to right, starting at 1.
SWORD	<i>fCType</i>	Input	<p>The C data type of the result data, which must be one of the following values:</p> <p>SQL_C_BINARY            SQL_C_BIT            SQL_C_CHAR            SQL_C_DATE            SQL_C_DEFAULT            SQL_C_DOUBLE            SQL_C_FLOAT            SQL_C_SLONG            SQL_C_SSHORT            SQL_C_STINYINT            SQL_C_TIME            SQL_C_TIMESTAMP            SQL_C_ULONG            SQL_C_USHORT            SQL_C_UTINYINT</p> <p>SQL_C_DEFAULT specifies that data be transferred to its default C data type.</p> <p>For information about how data is converted, see <a href="#">“Converting Data from SQL to C Data Types”</a> on page C-13.</p>
PTR	<i>rgbValue</i>	Input	<p>Pointer to storage for the data. If <i>rgbValue</i> is a null pointer, the driver unbinds the column. (To unbind all columns, an application calls <b>SQLFreeStmt</b> with the SQL_UNBIND option.)</p>

(1 of 2)

Type	Argument	Use	Description
SDWORD	<i>cbValueMax</i>	Input	Maximum length of the <i>rgbValue</i> buffer. For character data, <i>rgbValue</i> must also include space for the null-termination byte. For more information about length, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a> .
SDWORD FAR *	<i>pcbValue</i>	Input	<p>SQL_NULL_DATA or the number of bytes (excluding the null-termination byte for character data) available to return in <i>rgbValue</i> prior to calling <b>SQLExtendedFetch</b> or <b>SQLFetch</b>, or SQL_NO_TOTAL if the number of available bytes cannot be determined.</p> <p>For character data, if the number of bytes available to return is SQL_NO_TOTAL or is greater than or equal to <i>cbValueMax</i>, the data in <i>rgbValue</i> is truncated to <i>cbValueMax</i> – 1 bytes and is null-terminated by the driver.</p> <p>For binary data, if the number of bytes available to return is SQL_NO_TOTAL or is greater than <i>cbValueMax</i>, the data in <i>rgbValue</i> is truncated to <i>cbValueMax</i> bytes.</p> <p>For all other data types, the value of <i>cbValueMax</i> is ignored, and the driver assumes the size of <i>rgbValue</i> is the size of the C data type specified with <i>fCType</i>.</p> <p>For more information about the value returned in <i>pcbValue</i> for each <i>fCType</i>, see <a href="#">“Converting Data from SQL to C Data Types” on page C-13</a>.</p>

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or  
SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLBindCol** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values that **SQLBindCol** commonly returns and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	<b>INFORMIX-CLI</b> informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.
S1002	Invalid column number	The value specified for the argument <i>icol</i> exceeded the maximum number of columns supported by the data source.
S1003	Program type out of range	(DM) The argument <i>fCType</i> was not a valid data type or <code>SQL_C_DEFAULT</code> .
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for the argument <i>cbValueMax</i> was less than 0.
S1C00	Driver not capable	The driver does not support the data type specified in the argument <i>fCType</i> .  The argument <i>icol</i> was 0, and the driver does not support bookmarks.

## Usage

The INFORMIX-CLI interface provides the following ways to retrieve a column of data:

- **SQLBindCol** assigns the storage location for a column of data before the data is retrieved. When **SQLFetch** or **SQLExtendedFetch** is called, the driver places the data for all bound columns in the assigned locations.
- **SQLGetData** (an extended function) assigns a storage location for a column of data after **SQLFetch** or **SQLExtendedFetch** is called. It also places the data for the requested column in the assigned location. Because it can retrieve data from a column in parts, **SQLGetData** can retrieve long data values.

An application might choose to bind every column with **SQLBindCol**, to retrieve data only (and not bind) with **SQLGetData**, or to use a combination. However, unless the driver provides extended functionality, **SQLGetData** can be used only to retrieve data from columns that occur after the last bound column.

An application calls **SQLBindCol** to pass the pointer to the storage buffer for a column of data to the driver and to specify how or if to convert the data. The application must allocate enough storage for the data. If the buffer contains variable-length data, the application must allocate as much storage as the maximum length of the bound column, or the data might be truncated. For a list of valid data conversion types, see [“Converting Data from SQL to C Data Types” on page C-13](#).

At fetch time, the driver processes the data for each bound column according to the arguments specified in **SQLBindCol**. First, it converts the data according to the argument *fCType*. Next, it fills the buffer to which *rgbValue* points. Finally, it stores the available number of bytes in *pcbValue*; The value stored in *pcbValue* is the number of bytes available before the driver calls **SQLFetch** or **SQLExtendedFetch**.



Before the driver calls **SQLFetch** or **SQLExtendedFetch**, it returns a value, according to the following conditions:

- If `SQL_MAX_LENGTH` has been specified with **SQLSetStmtOption** and the available number of bytes is greater than `SQL_MAX_LENGTH`, the driver stores `SQL_MAX_LENGTH` in *pcbValue*.
- If the data is truncated because of `SQL_MAX_LENGTH`, but the user's buffer is large enough for `SQL_MAX_LENGTH` bytes of data, the driver returns `SQL_SUCCESS`.



**Important:** *The `SQL_MAX_LENGTH` statement option is intended to reduce network traffic. To guarantee that data is truncated, an application should allocate a buffer of the desired size and specify this size in the *cbValueMax* argument.*

- If the user's buffer causes the truncation, the driver returns `SQL_SUCCESS_WITH_INFO` and `SQLSTATE 01004` (Data truncated) for the fetch function.
- If the data value for a column is `NULL`, the driver sets *pcbValue* to `SQL_NULL_DATA`.
- If the number of bytes available to return cannot be determined in advance, the driver sets *pcbValue* to `SQL_NO_TOTAL`.

When an application uses **SQLExtendedFetch** to retrieve more than one row of data, it needs to call **SQLBindCol** only once for each column of the result set (in the same way that it binds a column in order to retrieve a single row of data with **SQLFetch**). The **SQLExtendedFetch** function coordinates placing each row of data into subsequent locations in the rowset buffers. For additional information about binding rowset buffers, see [“Usage” on page 13-131](#).

An application can call **SQLBindCol** to bind a column to a new storage location, regardless of whether data has already been fetched. The new binding replaces the old binding. The new binding does not apply to data already fetched; it takes effect the next time **SQLFetch** or **SQLExtendedFetch** is called.

To unbind a single bound column, an application calls **SQLBindCol** and specifies a null pointer for *rgbValue*; if *rgbValue* is a null pointer and the column is not bound, **SQLBindCol** returns `SQL_SUCCESS`. To unbind all bound columns, an application calls **SQLFreeStmt** with the `SQL_UNBIND` option.

## Code Example

In the following example, an application executes a `SELECT` statement to return a result set of the employee names, ages, and birthdays, which is sorted by birthday. It then calls `SQLBindCol` to bind the columns of data to local storage locations. Finally, the application fetches each row of data with `SQLFetch` and prints the name, age, and birthday of each employee.

For more code examples, see `SQLColumns` and `SQLExtendedFetch`.

```
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR  szName[NAME_LEN], szBirthday[BDAY_LEN];
SWORD  sAge;
SDWORD cbName, cbAge, cbBirthday;

retcode = SQLExecDirect(hstmt,
    "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE ORDER BY 3, 2, 1",
    SQL_NTS);

if (retcode == SQL_SUCCESS) {

    /* Bind columns 1, 2, and 3 */

    SQLBindCol(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
    SQLBindCol(hstmt, 2, SQL_C_SSHORT, &sAge, 0, &cbAge);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, szBirthday, BDAY_LEN, &cbBirthday);

    /* Fetch and print each row of data. On */
    /* an error, display a message and exit. */

    while (TRUE) {
        retcode = SQLFetch(hstmt);
        if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
            show_error();
        }
        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
            fprintf(out, "%-*s %-2d %*s", NAME_LEN-1, szName,
                sAge, BDAY_LEN-1, szBirthday);

            } else {
                break;
            }
        }
    }
}
```

## Related Functions

For information about	See
Returning information about a column in a result set	<b>SQLDescribeCol</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Freeing a statement handle	<b>SQLFreeStmt</b>
Fetching part or all of a column of data	<b>SQLGetData</b>
Returning the number of result set columns	<b>SQLNumResultCols</b>

## SQLBindParameter

Level 1

**SQLBindParameter** binds a buffer to a parameter marker in an SQL statement.

**SQLBindParameter** is an ODBC 2.0 function that replaces the ODBC 1.0 function **SQLSetParam**. For more information, see [“Usage” on page 13-28](#).

### Syntax

```
RETCODE SQLBIND (hstmt, ipar, fParamType, fCType, FSqlType,
cbColDef, ibScale, rgbValue, cbValueMax, pcbValue)
```

The **SQLBindParameter** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
UWORD	<i>ipar</i>	Input	Parameter number, ordered sequentially left to right, starting at 1
SWORD	<i>fParamType</i>	Input	The type of the parameter. For more information, see <a href="#">“fParamType Argument” on page 13-28.</a>
SWORD	<i>fCType</i>	Input	The C data type of the parameter. For more information, see <a href="#">“fCType Argument” on page 13-29.</a>
SWORD	<i>fSqlType</i>	Input	The SQL data type of the parameter. For more information, see <a href="#">“fSqlType Argument” on page 13-30.</a>
UDWORD	<i>cbColDef</i>	Input	The precision of the column or expression of the corresponding parameter marker. For more information, see <a href="#">“cbColDef Argument” on page 13-31.</a>
SWORD	<i>ibScale</i>	Input	The scale of the column or expression of the corresponding parameter marker. For further information concerning scale, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8.</a>

(1 of 2)

Type	Argument	Use	Description
PTR	<i>rgbValue</i>	Input/ Output	A pointer to a buffer for the data of the parameter. For more information, see <a href="#">“rgbValue Argument” on page 13-31.</a>
SDWORD	<i>cbValueMax</i>	Input	Maximum length of the <i>rgbValue</i> buffer. For more information, see <a href="#">“cbValueMax Argument” on page 13-32.</a>
SDWORD FAR *	<i>pcbValue</i>	Input/ Output	A pointer to a buffer for the length of the parameter. For more information, see <a href="#">“pcbValue Argument” on page 13-33.</a>

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLBindParameter** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLBindParameter** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value identified by the <i>fCType</i> argument cannot be converted to the data type identified by the <i>fSqlType</i> argument.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.
S1003	Program type out of range	(DM) The value specified by the argument <i>fCType</i> was not a valid data type or SQL_C_DEFAULT.
S1004	SQL data type out of range	(DM) The value specified for the argument <i>fSqlType</i> was in the block of numbers reserved for ODBC SQL data type indicators but was not a valid ODBC SQL data type indicator.
S1009	Invalid argument value	(DM) The argument <i>rgbValue</i> was a null pointer, the argument <i>pcbValue</i> was a null pointer, and the argument <i>iParamType</i> was not SQL_PARAM_OUTPUT.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for the argument <i>cbValueMax</i> was less than 0.

(1 of 2)

SQLSTATE	Error	Description
S1093	Invalid parameter number	<p>(DM) The value specified for the argument <i>ipar</i> was less than 1.</p> <p>The value specified for the argument <i>ipar</i> was greater than the maximum number of parameters supported by the data source.</p>
S1094	Invalid scale value	The value specified for the argument <i>ibScale</i> was outside the range of values supported by the data source for a column of the SQL data type specified by the <i>fSqlType</i> argument.
S1104	Invalid precision value	The value specified for the argument <i>cbColDef</i> was outside the range of values supported by the data source for a column of the SQL data type specified by the <i>fSqlType</i> argument.
S1105	Invalid parameter type	<p>(DM) The value specified for the argument <i>fParamType</i> was invalid. For more information, see <a href="#">“fParamType Argument” on page 13-28</a>.</p> <p>The value specified for the argument <i>fParamType</i> was SQL_PARAM_OUTPUT, and the parameter did not mark a return value from a procedure or a procedure parameter.</p> <p>The value specified for the argument <i>fParamType</i> was SQL_PARAM_INPUT, and the parameter marked the return value from a procedure.</p>
S1C00	Driver not capable	<p>The driver or data source does not support the conversion specified by the combination of the value specified for the argument <i>fCType</i> and the driver-specific value specified for the argument <i>fSqlType</i>.</p> <p>The driver or data source does not support the value specified for the argument <i>fSqlType</i>.</p>

(2 of 2)

## Usage

An application calls **SQLBindParameter** to bind each parameter marker in an SQL statement. Bindings are effective until the application calls **SQLBindParameter** again or until the application calls **SQLFreeStmt** with the `SQL_DROP` or `SQL_RESET_PARAMS` option.

For more information concerning parameter data types and parameter markers, see [“Parameter Data Types” on page B-2](#) and [“Parameter Markers” on page B-3](#).

### *fParamType* Argument

The *fParamType* argument specifies the parameter type. All parameters in SQL statements that do not call procedures, such as `INSERT` statements, are input parameters. Parameters in procedure calls can be input, input/output, or output parameters.

The *fParamType* argument is one of the following values:

- `SQL_PARAM_INPUT`

This parameter marks a parameter in an SQL statement that does not call a procedure, such as an `INSERT` statement, or it marks an input parameter in a procedure; these are collectively known as *input parameters*. For example, the parameters in `INSERT INTO Employee VALUES (?, ?, ?)` and `{call AddEmp(?, ?, ?)}` are input parameters.

When the statement executes, the driver sends data for the parameter to the data source; the *rgbValue* buffer must contain a valid input value or the *pcbValue* buffer must contain `SQL_NULL_DATA`, `SQL_DATA_AT_EXEC`, or the result of the `SQL_LEN_DATA_AT_EXEC` macro.

If an application cannot determine the type of a parameter in a procedure call, it sets *fParamType* to `SQL_PARAM_INPUT`; if the data source returns a value for the parameter, the driver discards it.



- SQL\_PARAM\_INPUT\_OUTPUT

The parameter marks an input/output parameter in a procedure. For example, the parameter in **{call GetEmpDept(?)}** is an input/output parameter that accepts the name of an employee and returns the department name of the employee.

When the statement executes, the driver sends data for the parameter to the data source; the *rgbValue* buffer must contain a valid input value or the *pcbValue* buffer must contain SQL\_NULL\_DATA, SQL\_DATA\_AT\_EXEC, or the result of the SQL\_LEN\_DATA\_AT\_EXEC macro. After the statement executes, the driver returns data for the parameter to the application; if the data source does not return a value for an input/output parameter, the driver sets the *pcbValue* buffer to SQL\_NULL\_DATA.

If an application calls **SQLSetParam**, the driver manager converts this to a call to **SQLBindParameter** in which the *fParamType* argument is set to SQL\_PARAM\_INPUT\_OUTPUT.

- SQL\_PARAM\_OUTPUT

The parameter marks the return value of a procedure or an output parameter in a procedure; these are collectively known as *output parameters*. For example, the parameter in **{?=call GetNextEmpID}** is an output parameter that returns the next employee ID.

After the statement executes, the driver returns data for the parameter to the application, unless the *rgbValue* and *pcbValue* arguments are both null pointers, in which case the driver discards the output value. If the data source does not return a value for an output parameter, the driver sets the *pcbValue* buffer to SQL\_NULL\_DATA.

### *fCType* Argument

The *fCType* argument is the C data type of the parameter. The *fCType* argument must be one of the following values:

- SQL\_C\_BINARY
- SQL\_C\_CHAR
- SQL\_C\_DATE
- SQL\_C\_DEFAULT
- SQL\_C\_DOUBLE

- SQL\_C\_FLOAT
- SQL\_C\_SLONG
- SQL\_C\_SSHORT
- SQL\_C\_TIME
- SQL\_C\_TIMESTAMP
- SQL\_C\_ULONG
- SQL\_C\_USHORT

SQL\_C\_DEFAULT specifies that the parameter value be transferred from the default C data type for the SQL data type specified with *fSqlType*.

For more information, see [“Default C Data Types”](#) on page C-6, [“Converting Data from C to SQL Data Types”](#) on page C-24, and [“Converting Data from SQL to C Data Types”](#) on page C-13.



### ***fSqlType* Argument**

The *fSqlType* argument must be one of the following values:

- SQL\_CHAR
- SQL\_DATE
- SQL\_DECIMAL
- SQL\_DOUBLE
- SQL\_INTEGER
- SQL\_LONGVARBINARY
- SQL\_LONGVARCHAR
- SQL\_REAL
- SQL\_SMALLINT
- SQL\_TIMESTAMP
- SQL\_VARCHAR

Values greater than SQL\_TYPE\_DRIVER\_START are reserved by ODBC. For information about how data is converted, see [“Converting Data from C to SQL Data Types”](#) on page C-24 and [“Converting Data from SQL to C Data Types”](#) on page C-13.

### ***cbColDef* Argument**

The *cbColDef* argument specifies the precision of the column or expression that corresponds to the parameter marker, unless all of the following conditions are true:

- An application calls **SQLSetParam**. The driver manager converts these calls to **SQLBindParameter**.
- The *fSqlType* argument is `SQL_LONGVARIABLE` or `SQL_LONGVARCHAR`.
- The data for the parameter is sent with **SQLPutData**.

In this case, the *cbColDef* argument contains the total number of bytes that are sent for the parameter. For more information, see [“Passing Parameter Values” on page 13-34](#).

### ***rgbValue* Argument**

The *rgbValue* argument points to a buffer that, when **SQLExecute** or **SQLExecDirect** is called, contains the actual data for the parameter. The data must be in the form specified by the *fCType* argument.

If *pcbValue* is the result of the `SQL_LEN_DATA_AT_EXEC(length)` macro or `SQL_DATA_AT_EXEC`, then *rgbValue* is an application-defined 32-bit value that is associated with the parameter. It is returned to the application through **SQLParamData**. For example, *rgbValue* might be a token such as a parameter number, a pointer to data, or a pointer to a structure that the application used to bind input parameters. If the parameter is an input/output parameter, *rgbValue* must be a pointer to a buffer where the output value is stored.

If **SQLParamOptions** is called to specify multiple values for the parameter, the application can use the value of the *pirow* argument in **SQLParamOptions** with the *rgbValue*. For example, *rgbValue* might point to an array of values and the application might use *pirow* to retrieve the correct value from the array. For more information, see [“Passing Parameter Values” on page 13-34](#).

If the *fParamType* argument is `SQL_PARAM_INPUT_OUTPUT` or `SQL_PARAM_OUTPUT`, *rgbValue* points to a buffer in which the driver returns the output value. If the procedure returns one or more result sets, the *rgbValue* buffer is not guaranteed to be set until all results have been fetched. If *fParamType* is `SQL_PARAM_OUTPUT` and *rgbValue* and *pcbValue* are both null pointers, the driver discards the output value.

If the application calls **SQLParamOptions** to specify multiple values for each parameter, *rgbValue* must point to an array. A single SQL statement processes the entire array of input values for an input or input/output parameter and returns an array of output values for an input/output or output parameter.

### ***cbValueMax* Argument**

For character and binary C data, the *cbValueMax* argument specifies the length of the *rgbValue* buffer, if it is a single element, or the length of an element in the *rgbValue* array if the application calls **SQLParamOptions** to specify multiple values for each parameter. If the application specifies multiple values, *cbValueMax* determines the location of values in the *rgbValue* array, both on input and on output. For input/output and output parameters, it determines whether to truncate character and binary C data on output.

For character C data, if the number of bytes available to return is greater than or equal to *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* - 1 bytes and is null-terminated by the driver.

For binary C data, if the number of bytes available to return is greater than *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* bytes.

For all other types of C data, the *cbValueMax* argument is ignored. The length of the *rgbValue* buffer (if it is a single element) or the length of an element in the *rgbValue* array (if the application calls **SQLParamOptions** to specify multiple values for each parameter) is assumed to be the length of the C data type.

When an ODBC 1.0 application calls **SQLSetParam** in an ODBC 2.x driver, the driver manager converts this to a call to **SQLBindParameter** in which the *cbValueMax* argument is always `SQL_SETPARAM_VALUE_MAX`. Because the driver manager returns an error when an ODBC 2.x application sets *cbValueMax* to `SQL_SETPARAM_VALUE_MAX`, an ODBC 2.x driver can use this to determine when it is called by an ODBC 1.0 application.

In **SQLSetParam**, the driver defines how an application specifies the length of the *rgbValue* buffer, so that the driver can return character or binary data, and how an application sends an array of character or binary parameter values to the driver.

### ***pcbValue* Argument**

The *pcbValue* argument points to a buffer that, when **SQLExecute** or **SQLExecDirect** is called, contains one of the following values:

- The length of the parameter value stored in *rgbValue*  
This is ignored except for character or binary C data.
- **SQL\_NTS**  
The parameter value is a null-terminated string.
- **SQL\_NULL\_DATA**  
The parameter value is NULL.
- **SQL\_DEFAULT\_PARAM**  
A procedure is to use the default value of a parameter, rather than a value retrieved from the application. This value is valid only in a procedure call and then only if the *fParamType* argument is **SQL\_PARAM\_INPUT** or **SQL\_PARAM\_INPUT\_OUTPUT**. When *pcbValue* is **SQL\_DEFAULT\_PARAM**, the corresponding parameter can only be a parameter for an ODBC canonical procedure invocation. When *pcbValue* is **SQL\_DEFAULT\_PARAM**, the *fCType*, *fSqlType*, *cbColDef*, *ibScale*, *cbValueMax* and *rgbValue* arguments are ignored for input parameters and are used only to define the output parameter value for input/output parameters. This value was introduced in ODBC 2.0.
- The result of the **SQL\_LEN\_DATA\_AT\_EXEC(length)** macro  
The data for the parameter is sent with **SQLPutData**. If the *fSqlType* argument is **SQL\_LONGVARBINARY**, **SQL\_LONGVARCHAR**, or a long, data-source-specific data type and the driver returns “Y” for the **SQL\_NEED\_LONG\_DATA\_LEN** information type in **SQLGetInfo**, length is the number of bytes of data to be sent for the parameter; otherwise, length must be a nonnegative value and is ignored. For more information, see [“Passing Parameter Values” on page 13-34](#).  
For example, to send 10,000 bytes of data with **SQLPutData** for an **SQL\_LONGVARCHAR** parameter, an application sets *pcbValue* to **SQL\_LEN\_DATA\_AT\_EXEC(10000)**. This macro was introduced in ODBC 2.0.



If *pcbValue* is a null pointer, the driver assumes that all input parameter values are non-null and that character and binary data are null-terminated. If *fParamType* is `SQL_PARAM_OUTPUT` and *rgbValue* and *pcbValue* are both null pointers, the driver discards the output value.

**Important:** Application developers are strongly discouraged from specifying a null pointer for *pcbValue* when the data type of the parameter is `SQL_C_BINARY`. For `SQL_C_BINARY` data, the driver sends only the data preceding an occurrence of the null-termination character, `0x00`. To ensure that the driver does not unexpectedly truncate `SQL_C_BINARY` data, *pcbValue* should contain a pointer to a valid length value.

If the *fParamType* argument is `SQL_PARAM_INPUT_OUTPUT` or `SQL_PARAM_OUTPUT`, *pcbValue* points to a buffer in which the driver returns `SQL_NULL_DATA`, the number of bytes available to return in *rgbValue* (excluding the null-termination byte of character data), or `SQL_NO_TOTAL` if the number of bytes available to return cannot be determined. If the procedure returns one or more result sets, the *pcbValue* buffer is not guaranteed to be set until all the results have been fetched.

If the application calls **SQLParamOptions** to specify multiple values for each parameter, *pcbValue* points to an array of `SDWORD` values. These values can be any of the values listed earlier in this section and are processed with a single SQL statement.

### **Passing Parameter Values**

An application can pass the value for a parameter either in the *rgbValue* buffer or with one or more calls to **SQLPutData**. Parameters whose data is passed with **SQLPutData** are known as *data-at-execution* parameters. These are commonly used to send data for `SQL_LONGVARBINARY` and `SQL_LONGVARCHAR` parameters and can be mixed with other parameters.

### To pass parameter values

1. The application calls **SQLBindParameter** for each parameter to bind buffers for the value (*rgbValue* argument) and length (*pcbValue* argument) of the parameter. For data-at-execution parameters, *rgbValue* is an application-defined 32-bit value such as a parameter number or a pointer to data. The value is returned later and can be used to identify the parameter.
2. The application places values for input and input/output parameters in the *rgbValue* and *pcbValue* buffers.
  - For normal parameters, the application places the parameter value in the *rgbValue* buffer and the length of that value in the *pcbValue* buffer.
  - For data-at-execution parameters, the application places the result of the `SQL_LEN_DATA_AT_EXEC(length)` macro in the *pcbValue* buffer.
3. The application calls **SQLExecute** or **SQLExecDirect** to execute the SQL statement.
  - If no data-at-execution parameters exist, the process is complete.
  - If any data-at-execution parameters exist, the function returns `SQL_NEED_DATA`.
4. The application calls **SQLParamData** to retrieve the application-defined value specified in the *rgbValue* argument for the first data-at-execution parameter to be processed.

The data-at-execution parameters are similar to data-at-execution columns, although the value that **SQLParamData** returns is different for each.
5. The application calls **SQLPutData** one or more times to send data for the parameter. More than one call is needed if the data value is larger than the *rgbValue* buffer specified in **SQLPutData**; multiple calls to **SQLPutData** for the same parameter are allowed only when sending character C data to a column with a character, binary, or data-source-specific data type or when sending binary C data to a column with a character, binary, or data-source-specific data type.

6. The application calls **SQLParamData** again to signal that all data has been sent for the parameter.
  - If more data-at-execution parameters exist, **SQLParamData** returns `SQL_NEED_DATA` and the application-defined value for the next data-at-execution parameter to be processed. The application repeats steps 5 and 6.
  - If no more data-at-execution parameters exist, the process is complete. If the statement executes successfully, **SQLParamData** returns `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`; if the execution fails, it returns `SQL_ERROR`. At this point, **SQLParamData** can return any `SQLSTATE` that can be returned by the function used to execute the statement (**SQLExecDirect** or **SQLExecute**).

Output values for any input/output or output parameters are available in the *rgbValue* and *pcbValue* buffers after the application retrieves any result sets that the statement generates.

If after **SQLExecute** or **SQLExecDirect** returns `SQL_NEED_DATA` and before data is sent for all data-at-execution parameters, the statement is canceled or an error occurs in **SQLParamData** or **SQLPutData**, the application can call only **SQLCancel**, **SQLGetFunctions**, **SQLParamData**, or **SQLPutData** with the *hstmt* or the *hdbc* associated with the *hstmt*. If the application calls any other function with the *hstmt* or the *hdbc* associated with the *hstmt*, the function returns `SQL_ERROR` and `SQLSTATE S1010` (Function-sequence error).

If the application calls **SQLCancel** while the driver still needs data for data-at-execution parameters, the driver stops executing the statement; the application can then call **SQLExecute** or **SQLExecDirect** again. If the application calls **SQLParamData** or **SQLPutData** after it cancels the statement, the function returns `SQL_ERROR` and `SQLSTATE S1008` (Operation canceled).



## Code Example

In the following example, an application prepares an SQL statement to insert data into the **EMPLOYEE** table. The SQL statement contains parameters for the **NAME**, **AGE**, and **BIRTHDAY** columns. For each parameter in the statement, the application calls **SQLBindParameter** to specify the ODBC C data type and the SQL data type of the parameter and to bind a buffer to each parameter. For each row of data, the application assigns data values to each parameter and calls **SQLExecute** to execute the statement.

```
#define NAME_LEN 30

UCHAR      szName[NAME_LEN];
SWORD      sAge;
SDWORD     cbName = SQL_NTS, cbAge = 0, cbBirthday = 0;
DATE_STRUCT dsBirthday;

retcode = SQLPrepare(hstmt,
    "INSERT INTO EMPLOYEE (NAME, AGE, BIRTHDAY) VALUES (?, ?, ?)",
    SQL_NTS);

if (retcode == SQL_SUCCESS) {

    /* Specify data types and buffers.          */
    /* for Name, Age, Birthday parameter data. */

    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
        SQL_CHAR, NAME_LEN, 0, szName, 0, &cbName);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT,
        SQL_SMALLINT, 0, 0, &sAge, 0, &cbAge);
    SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_DATE,
        SQL_DATE, 0, 0, &dsBirthday, 0, &cbBirthday);
    strcpy(szName, "Smith, John D."); /* Specify first row of */
    sAge = 40; /* parameter data */
    dsBirthday.year = 1952;
    dsBirthday.month = 2;
    dsBirthday.day = 29;
    retcode = SQLExecute(hstmt); /* Execute statement with */
    /* first row */

    strcpy(szName, "Jones, Bob K."); /* Specify second row of */
    sAge = 52; /* parameter data */
    dsBirthday.year = 1940;
    dsBirthday.month = 3;
    dsBirthday.day = 31;
    SQLExecute(hstmt); /* Execute statement with */
    /* second row */

}

```

For other similar examples, see **SQLParamOptions**, **SQLProcedures**, and **SQLPutData**.

## Related Functions

For information about	See
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Returning the number of statement parameters	<b>SQLNumParams</b>
Returning the next parameter to send data for	<b>SQLParamData</b>
Specifying multiple parameter values	<b>SQLParamOptions</b>
Sending parameter data at execution time	<b>SQLPutData</b>

## SQLBrowseConnect

### Level 2

**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 are enumerated, a connection to the data source is completed, and **SQLBrowseConnect** returns a complete connection string. A return code of `SQL_SUCCESS_WITH_INFO` or `SQL_SUCCESS` indicates that all connection information has been specified and the application is now connected to the data source.

## Syntax

```
RETCODE SQLBrowseConnect(hdbc, szConnStrIn, cbConnStrIn,
szConnStrOut, cbConnStrOutMax, pcbConnStrOut)
```

The **SQLBrowseConnect** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
UCHAR FAR *	<i>szConnStrIn</i>	Input	Browse request connection string. (See “ <a href="#">szConnStrIn Argument</a> ” on page 13-43.)
SWORD	<i>cbConnStrIn</i>	Input	Length of <i>szConnStrIn</i> .
UCHAR FAR *	<i>szConnStrOut</i>	Output	Pointer to storage for the browse result connection string. (See “ <a href="#">szConnStrOut Argument</a> ” on page 13-43.)
SWORD	<i>cbConnStrOutMax</i>	Input	Maximum length of the <i>szConnStrOut</i> buffer.
SWORD FAR *	<i>pcbConnStrOut</i>	Output	The total number of bytes (excluding the null-termination byte) available to return in <i>szConnStrOut</i> . If the number of bytes available to return is greater than or equal to <i>cbConnStrOutMax</i> , the connection string in <i>szConnStrOut</i> is truncated to <i>cbConnStrOutMax</i> – 1 bytes.

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NEED\_DATA, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLBrowseConnect** returns `SQL_ERROR`, `SQL_SUCCESS_WITH_INFO`, or `SQL_NEED_DATA`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLBrowseConnect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The buffer <i>szConnStrOut</i> was not large enough to return entire browse result connection string, so the string was truncated. The argument <i>pcbConnStrOut</i> 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 ( <i>szConnStrIn</i> ). (Function returns <code>SQL_NEED_DATA</code> .)  An attribute keyword was specified in the browse request connection string ( <i>szConnStrIn</i> ) that does not apply to the current connection level (function returns <code>SQL_NEED_DATA</code> ).
08001	Unable to connect to data source	The driver could not establish a connection with the data source.
08002	Connection in use	(DM) The specified <i>hdbc</i> already established a connection with a data source, and the connection is open.
08004	Data source rejected establishment of connection	The data source rejected the establishment of the connection for implementation-defined reasons.

(1 of 3)

SQLSTATE	Error	Description
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
28000	Invalid authorization specification	Either the user identifier or the authorization string or both as specified in the browse-request connection string ( <i>szConnStrIn</i> ) violated restrictions defined by the data source.
IM002	Data source not found and no default driver specified	(DM) INFORMIX-CLI cannot find the data-source name specified in the browse-request connection string ( <i>szConnStrIn</i> ) in the <b>odbc.ini</b> file, and a default driver specification does not exist.  (DM) INFORMIX-CLI cannot find the <b>odbc.ini</b> file.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the <b>odbc.ini</b> file, specified by the DRIVER keyword, was not found or could not be loaded for some other reason.
IM004	Driver <b>SQLAllocEnv</b> failed	(DM) During <b>SQLBrowseConnect</b> , the driver manager called the driver <b>SQLAllocEnv</b> function, and the driver returned an error.
IM005	Driver <b>SQLAllocConnect</b> failed	(DM) During <b>SQLBrowseConnect</b> , the driver manager called the driver <b>SQLAllocConnect</b> function and the driver returned an error.
IM006	Driver <b>SQLSetConnectOption</b> failed	(DM) During <b>SQLBrowseConnect</b> , the driver manager called the driver <b>SQLSetConnectOption</b> function, and the driver returned an error.
IM009	Unable to load translation-shared library	The driver did not load the translation-shared library that was specified for the data source or for the connection.
IM010	Data-source name too long	(DM) The attribute value for the DSN keyword was longer than <b>SQL_MAX_DSN_LENGTH</b> characters.

(2 of 3)

SQLSTATE	Error	Description
IM011	Driver name too long	(DM) The attribute value for the DRIVER keyword was longer than 255 characters.
IM012	DRIVER keyword syntax error	(DM) The keyword-value pair for the DRIVER keyword contained a syntax error.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	(DM) The driver manager did not allocate memory required to support executing or completing the function.  The driver did not allocate memory required to support executing or completing the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbConnStrIn</i> was less than 0 and was not equal to SQL_NTS.  (DM) The value specified for argument <i>cbConnStrOutMax</i> was less than 0.
S1T00	Time-out expired	The time-out period expired before the connection to the data source completed. The time-out period is set through <b>SQLSetConnectOption</b> , SQL_LOGIN_TIMEOUT.

(3 of 3)

## Usage

Each time an application calls **SQLBrowseConnect**, the function validates the current attributes, returns the next level of attributes, and returns a user-friendly name for each attribute. It might also return a list of valid values for those attributes. After an application has specified each level of attributes and values, **SQLBrowseConnect** connects to the data source and returns a complete connection string. This string can be used with **SQLDriverConnect** to reconnect to the data source.

## ***szConnStrIn Argument***

A browse-request connection string has the following syntax:

```

connection-string ::= attribute[;] | attribute; connection-string
attribute ::= attribute-keyword=attribute-value | DRIVER={attribute-value}
(The braces are literal; the application must specify them.)
attribute-keyword ::= DSN | UID | PWD
                    | driver-defined-attribute-keyword
attribute-value ::= character-string
driver-defined-attribute-keyword ::= identifier

```

In this example, *character-string* has zero or more characters; *identifier* has one or more characters; *attribute-keyword* is case insensitive; *attribute-value* might be case sensitive; and the value of the DSN keyword does not consist solely of blanks. Because of connection-string and initialization-file grammar, avoid keywords and attribute values that contain the characters [ ] { } ( ) , ; ? \* = ! @ \ . Because of the registry grammar, keywords and data-source names cannot contain a backslash (\).

If any keywords are repeated in the browse-request connection string, the driver 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, the driver manager and driver use whichever keyword appears first.

## ***szConnStrOut 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 the driver.)
attribute-value-list ::= character-string | character-string, attribute-value-list

```

In this example, *character-string* has zero or more characters, *identifier* and *localized-identifier* have one or more characters, *attribute-keyword* is not case sensitive, and *attribute-value* might be case sensitive. Because of connection-string and initialization-file grammar, avoid keywords, localized identifiers, and attribute values that contain the characters [ ] { } ( ) , ; ? \* = ! @ \. Because of the registry grammar, keywords and data-source names cannot contain a backslash (\).

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**.
- A *driver-defined-attribute-keyword* names the kind of attribute for which an attribute value can be supplied. For example, it might be SERVER, DATABASE, HOST, or DBMS.
- *ODBC-attribute-keywords* and *driver-defined-attribute-keywords* include a localized or user-friendly version of the keyword. This localized version might be used by applications as a label in a dialog box. However, UID, PWD, or the *identifier* alone must be used when the application passes a browse-request connection string to the driver.
- The {*attribute-value-list*} is an enumeration of actual values valid for the corresponding *attribute-keyword*. The braces ({} ) do not indicate a list of choices; the driver returns them. For example, the list might include 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. The driver associates state information with the connection handle so that the context can be determined on each call.



## Using SQLBrowseConnect

**SQLBrowseConnect** requires an allocated *hdbc*. The driver manager loads the driver that is specified in or that corresponds to the data-source name specified in the initial browse-request connection string. For additional information, see [“Usage” on page 13-78](#). It might establish a connection with the data source during the browsing process. If **SQLBrowseConnect** returns `SQL_ERROR`, outstanding connections terminate, and the *hdbc* returns to an unconnected state.

When **SQLBrowseConnect** is called for the first time on an *hdbc*, the browse-request connection string must contain the DSN keyword or the DRIVER keyword. If the browse-request connection string contains the DSN keyword, the driver manager locates a corresponding data-source specification in the **odbc.ini** file, as the following list describes:

- If the driver manager finds the corresponding data-source specification, it loads the associated driver shared library; the driver can retrieve information about the data source from the **odbc.ini** file.
- If the driver manager cannot find the corresponding data-source specification, it locates the default data-source specification and loads the associated driver shared library; the driver can retrieve information about the default data source from the **odbc.ini** file.
- If the driver manager cannot find the corresponding data-source specification, and no default data-source specification exists, it returns `SQL_ERROR` with `SQLSTATE IM002` (Data source not found and no default driver specified).

If the browse-request connection string contains the DRIVER keyword, the driver manager loads the specified driver; it does not attempt to locate a data source in the **odbc.ini** file. Because the DRIVER keyword does not use information from the **odbc.ini** file, the driver must define enough keywords so a driver can connect to a data source using only the information in the browse-request connection strings.

On each call to **SQLBrowseConnect**, the application specifies the connection attribute values in the browse-request connection string. The driver 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**. The application cannot use the contents of previous browse-result connection strings when it builds the current one; that is, it cannot specify different values for attributes set in previous levels.

When all levels of connection and their associated attributes are enumerated, the driver returns `SQL_SUCCESS`, the connection to the data source is complete, and a complete connection string returns to the application.

The connection string can be used with **SQLDriverConnect** with the `SQL_DRIVER_NOPROMPT` option to establish another connection.

**SQLBrowseConnect** also returns `SQL_NEED_DATA` if recoverable, nonfatal errors occur 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` returns and the browse-result connection string is unchanged, an error has occurred, and the application must call **SQLError** 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**. The driver terminates any outstanding connections and returns the *hdbc* to an unconnected state.

For more information, see [“Connection Browsing with SQLBrowseConnect” on page 5-9](#).

## Code Example

In the following example, an application calls **SQLBrowseConnect** repeatedly. Each time **SQLBrowseConnect** returns `SQL_NEED_DATA`, it passes back information about the data that it needs in *szConnStrOut*. The application passes *szConnStrOut* to its routine **GetUserInput** (not shown). **GetUserInput** parses the information, builds and displays a dialog box, and

returns the information entered by the user in *szConnStrIn*. The application passes the user's information to the driver in the next call to **SQLBrowseConnect**. After the application provides all the necessary information for the driver to connect to the data source, **SQLBrowseConnect** returns `SQL_SUCCESS`, and the application proceeds.

For example, to connect to the data source **My Source**, the following actions might occur. First, the application passes the following string to **SQLBrowseConnect**, as shown in the following example:

```
"DSN=My Source"
```

The driver manager loads the driver associated with the data source **My Source**. It then calls the driver **SQLBrowseConnect** function with the same arguments that it received from the application. The driver returns the following string in *szConnStrOut*:

```
"HOST:Server={red,blue,green};UID:ID=?;PWD>Password=?"
```

The application passes this string to its **GetUserInput** routine, which provides a dialog box that asks the user to select the red, blue, or green database server and to enter a user ID and password. The routine passes the following user-specified information back in *szConnStrIn*, which the application passes to **SQLBrowseConnect**:

```
"HOST=red;UID=Smith;PWD=Sesame"
```

**SQLBrowseConnect** uses this information to connect to the red database server as **Smith** with the password **Sesame**, and returns the following string in *szConnStrOut*:

```
"*DATABASE:Database={master,model,empdata}"
```

The application passes this string to its **GetUserInput** routine, which provides a dialog box that asks the user to select a database. The user selects **empdata**, and the application calls **SQLBrowseConnect** a final time with the following string:

```
"DATABASE=empdata"
```

This is the final piece of information that the driver needs to connect to the data source; **SQLBrowseConnect** returns **SQL\_SUCCESS**, and *szConnStrOut* contains the completed connection string.

```

"DSN=My Source;HOST=red;UID=Smith;PWD=Sesame;DATABASE=empdata"

#define BRWS_LEN 100
HENV    henv;
HDBC    hdbc;
HSTMT   hstmt;
RETCODE retcode;
UCHAR   szConnStrIn[BRWS_LEN], szConnStrOut[BRWS_LEN];
SWORD   cbConnStrOut;

retcode = SQLAllocEnv(&henv);           /* Environment handle */
if (retcode == SQL_SUCCESS) {
    retcode = SQLAllocConnect(henv, &hdbc); /* Connection handle */
    if (retcode == SQL_SUCCESS) { /* Call SQLBrowseConnect until it returns a
value other than */
        /* SQL_NEED_DATA (pass the data source name the first time). */
        /* If SQL_NEED_DATA is returned, call GetUserInput (not
        /* shown) to build a dialog from the values in szConnStrOut. */
        /* The user-supplied values are returned in szConnStrIn,
        /* which is passed in the next call to SQLBrowseConnect.
        /*

        strcpy(szConnStrIn, "DSN=MyServer");
        do {
            retcode = SQLBrowseConnect(hdbc, szConnStrIn, SQL_NTS,
                szConnStrOut, BRWS_LEN, &cbConnStrOut)
            if (retcode == SQL_NEED_DATA)
                GetUserInput(szConnStrOut, szConnStrIn);
        } while (retcode == SQL_NEED_DATA);

        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){

            /* Process data after successful connection */

            retcode = SQLAllocStmt(hdbc, &hstmt);
            if (retcode == SQL_SUCCESS) {
                ...;
                ...;
                ...;
                SQLFreeStmt(hstmt, SQL_DROP);
            }
            SQLDisconnect(hdbc);
        }
    }
    SQLFreeConnect(hdbc);
}
SQLFreeEnv(henv);

```

## Related Functions

For information about	See
Allocating a connection handle	<b>SQLAllocConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Disconnecting from a data source	<b>SQLDisconnect</b>
Connecting to a data source using a connection string or dialog box	<b>SQLDriverConnect</b>
Returning driver descriptions and attributes	<b>SQLDrivers</b>
Freeing a connection handle	<b>SQLFreeConnect</b>

## SQLCancel

**SQLCancel** cancels the processing on an *hstmt*.

### Syntax

```
RETCODE SQLCancel(hstmt)
```

The **SQLCancel** function accepts the following argument.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLCancel** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLCancel** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
70100	Operation aborted	The data source did not process the cancel request.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

## Usage

**SQLCancel** can cancel function processing on an *hstmt* that needs data.

If an application calls **SQLCancel** when processing does not involve the *hstmt*, **SQLCancel** has the same effect as **SQLFreeStmt** with the `SQL_CLOSE` option; this behavior is defined only for completeness, and applications should call **SQLFreeStmt** to close cursors.

### ***Canceling Functions that Need Data***

After **SQLExecute** or **SQLExecDirect** returns `SQL_NEED_DATA` and before data has been sent for all data-at-execution parameters, an application can call **SQLCancel** to cancel the statement execution. After the statement is canceled, the application can call **SQLExecute** or **SQLExecDirect** again. For more information, see [“Passing Parameter Values”](#) on page 13-34.

### ***Canceling Functions in Multithreaded Applications***

In a multithreaded application, the application can cancel a function that is running synchronously on an *hstmt*. To cancel the function, the application calls **SQLCancel** with the same *hstmt* as that used by the target function, but on a different thread. The return code of **SQLCancel** indicates whether or not the driver processed the request successfully. The return code of the original function indicates whether the function completed normally or was canceled.

### ***Related Functions***

<b>For information about</b>	<b>See</b>
Assigning storage for a parameter	<b>SQLBindParameter</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Freeing a statement handle	<b>SQLFreeStmt</b>
Returning the next parameter for which to send data	<b>SQLParamData</b>
Sending parameter data at execution time	<b>SQLPutData</b>

## SQLColAttributes

**SQLColAttributes** returns descriptor information for a column in a result set; it cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

### Syntax

```
RETCODE SQLColAttributes(hstmt, icol, fDescType, rgbDesc,
cbDescMax, pcbDesc, pfDesc)
```

The **SQLColAttributes** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>icol</i>	Input	Column number of result data, ordered sequentially from left to right, starting at 1. Columns can be described in any order.
UWORD	<i>fDescType</i>	Input	A valid descriptor type. (See <a href="#">“Usage” on page 13-55.</a> )
PTR	<i>rgbDesc</i>	Output	Pointer to storage for the descriptor information. The format of the descriptor information returned depends on the <i>fDescType</i> .
SWORD	<i>cbDescMax</i>	Input	Maximum length of the <i>rgbDesc</i> buffer.

(1 of 2)



Type	Argument	Use	Description
SWORD FAR *	<i>pcbDesc</i>	Output	<p>Total number of bytes (excluding the null-termination byte for character data) available to return in <i>rgbDesc</i>.</p> <p>For character data, if the number of bytes available to return is greater than or equal to <i>cbDescMax</i>, the descriptor information in <i>rgbDesc</i> is truncated to <i>cbDescMax</i> - 1 bytes and is null-terminated by the driver.</p> <p>For all other types of data, the value of <i>cbValueMax</i> is ignored and the driver assumes the size of <i>rgbValue</i> is 32 bits.</p>
SDWORD FAR *	<i>pfDesc</i>	Output	<p>Pointer to an integer value to contain descriptor information for numeric descriptor types, such as SQL_COLUMN_LENGTH.</p>

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLColAttributes** returns either SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLColAttributes** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer <i>rgbDesc</i> was not large enough to return the entire string value, so the string value was truncated. The argument <i>pcbDesc</i> contains the length of the untruncated string value (function returns SQL_SUCCESS_WITH_INFO).
24000	Invalid cursor state	The statement associated with the <i>hstmt</i> does not return a result set. There are no columns to describe.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.
S1002	Invalid column number	(DM) The value specified for the argument <i>icol</i> was 0, and the argument <i>iDescType</i> was not SQL_COLUMN_COUNT.  The value specified for the argument <i>icol</i> was greater than the number of columns in the result set, and the argument <i>iDescType</i> was not SQL_COLUMN_COUNT.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.

(1 of 2)

SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) The function was called prior to calling <b>SQLPrepare</b> or <b>SQLExecDirect</b> for the <i>hstmt</i> .  (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for the argument <i>cbDescMax</i> was less than 0.
S1091	Descriptor type out of range	(DM) The value specified for the argument <i>fDescType</i> was in the block of numbers reserved for ODBC descriptor types but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The driver or data source does not support the value specified for the argument <i>fDescType</i> .
S1T00	Time-out expired	The time-out period expired before the data source returned the requested information. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

**SQLColAttributes** can return any SQLSTATE that can be returned by **SQLPrepare** or **SQLExecute** when it is called after **SQLPrepare** and before **SQLExecute** depending on when the data source evaluates the SQL statement associated with the *hstmt*.

## Usage

**SQLColAttributes** returns information either in *pfDesc* or in *rgbDesc*. Integer information is returned in *pfDesc* as a 32-bit, signed value; all other formats of information are returned in *rgbDesc*. When information is returned in *pfDesc*, the driver ignores *rgbDesc*, *cbDescMax*, and *pcbDesc*. When information is returned in *rgbDesc*, the driver ignores *pfDesc*.

This function is an extensible alternative to **SQLDescribeCol**.

**SQLDescribeCol** returns a fixed set of descriptor information based on ANSI-89 SQL. **SQLColAttributes** allows access to the more extensive set of descriptor information available in ANSI SQL-92 and Informix extensions.

The following table shows the currently defined descriptor types and the arguments in which information is returned for them; more descriptor types will probably be defined to take advantage of different data sources.

If a descriptor type does not apply to a driver or data source, then, unless otherwise stated, the driver returns 0 in *pcbDesc* or an empty string in *rgbDesc*.

fDescType	Information Returned in	Description
SQL_COLUMN_AUTO_INCREMENT	<i>pfDesc</i>	TRUE if the column is auto increment.  FALSE if the column is not auto increment or is not numeric.  Auto increment is valid for numeric data type columns only. An application can insert values into an auto-increment column but cannot update values in the column.
SQL_COLUMN_CASE_SENSITIVE	<i>pfDesc</i>	TRUE if the column is treated as case sensitive for collations and comparisons.  FALSE if the column is not treated as case sensitive for collations and comparisons or is noncharacter.
SQL_COLUMN_COUNT	<i>pfDesc</i>	Number of columns available in the result set. The <i>icol</i> argument is ignored.
SQL_COLUMN_DISPLAY_SIZE	<i>pfDesc</i>	Maximum number of characters required to display data from the column. For more information on display size, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8.</a>

(1 of 4)

fDescType	Information Returned in	Description
SQL_COLUMN_LABEL	<i>rgbDesc</i>	<p>The column label or title. For example, a column named <b>EmpName</b> might be labeled Employee Name.</p> <p>If a column does not have a label, the column name is returned. If the column is unlabeled and unnamed, an empty string is returned.</p>
SQL_COLUMN_LENGTH	<i>pfDesc</i>	<p>The length in bytes of data transferred on an <b>SQLGetData</b> or <b>SQLFetch</b> operation if <b>SQL_C_DEFAULT</b> is specified. For numeric data, this size can be different than the size of the data stored on the data source. For more length information, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a>.</p>
SQL_COLUMN_MONEY	<i>pfDesc</i>	<p>TRUE if the column is MONEY data type.</p> <p>FALSE if the column is not MONEY data type.</p>
SQL_COLUMN_NAME	<i>rgbDesc</i>	<p>The column name.</p> <p>If the column is unnamed, an empty string is returned.</p>
SQL_COLUMN_NULLABLE	<i>pfDesc</i>	<p>SQL_NO_NULLS if the column does not accept NULL values.</p> <p>SQL_NULLABLE if the column accepts NULL values.</p> <p>SQL_NULLABLE_UNKNOWN if it is not known whether the column accepts NULL values.</p>
SQL_COLUMN_OWNER_NAME	<i>rgbDesc</i>	<p>The owner of the table that contains the column. The returned value is implementation defined if the column is an expression or if the column is part of a view. If the data source does not support owners or the owner name cannot be determined, an empty string is returned.</p>

(2 of 4)

fDescType	Information Returned in	Description
SQL_COLUMN_PRECISION	<i>pfDesc</i>	The precision of the column on the data source. For more information on precision, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a> .
SQL_COLUMN_QUALIFIER_NAME	<i>rgbDesc</i>	The qualifier of the table that contains the column. The returned value is implementation defined if the column is an expression or if the column is part of a view. If the data source does not support qualifiers or the qualifier name cannot be determined, an empty string is returned.
SQL_COLUMN_SCALE	<i>pfDesc</i>	The scale of the column on the data source. For more information on scale, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a> .
SQL_COLUMN_SEARCHABLE	<i>pfDesc</i>	<p>SQL_UNSEARCHABLE if the column cannot be used in a WHERE clause.</p> <p>SQL_LIKE_ONLY if the column can be used in a WHERE clause only with the LIKE predicate.</p> <p>SQL_ALL_EXCEPT_LIKE if the column can be used in a WHERE clause with all comparison operators except LIKE.</p> <p>SQL_SEARCHABLE if the column can be used in a WHERE clause with any comparison operator.</p>
SQL_COLUMN_TABLE_NAME	<i>rgbDesc</i>	<p>Columns of type SQL_LONGVARCHAR and SQL_LONGVARBINARY usually return SQL_LIKE_ONLY.</p> <p>The name of the table that contains the column. The returned value is implementation defined if the column is an expression or if the column is part of a view.</p> <p>If the table name cannot be determined, an empty string is returned.</p>

(3 of 4)

fDescType	Information Returned in	Description
SQL_COLUMN_TYPE	<i>pfDesc</i>	SQL data type. For a list of valid ODBC SQL data types, see <a href="#">Appendix C</a> . For information on how Informix data types map to ODBC SQL data types, see “ <a href="#">Mapping Data Types</a> ” on page 1-15
SQL_COLUMN_TYPE_NAME	<i>rgbDesc</i>	Data-source-dependent data type name; for example, CHAR, VARCHAR, MONEY, LONG VARBINARY, or CHAR ( ) FOR BIT DATA.  If the type is unknown, an empty string is returned.
SQL_COLUMN_UNSIGNED	<i>pfDesc</i>	TRUE if the column is unsigned or not numeric.  FALSE if the column is signed.
SQL_COLUMN_UPDATABLE	<i>pfDesc</i>	Column is described by the values for the defined constants:  SQL_ATTR_READONLY SQL_ATTR_WRITE SQL_ATTR_READWRITE_UNKNOWN  SQL_COLUMN_UPDATABLE describes the updatability of the column in the result set. 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_ATTR_READWRITE_UNKNOWN should be returned.

(4 of 4)

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning information about a column in a result set	<b>SQLDescribeCol</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>

## SQLColumnPrivileges

**SQLColumnPrivileges** returns a list of columns and associated privileges for the specified table. The driver returns the information as a result set on the specified *hstmt*.

### Syntax

```
RETCODE SQLColumnPrivileges(hstmt, szTableQualifier,
                             cbTableQualifier, szTableOwner, cbTableOwner, szTableName,
                             cbTableName, szColumnName, cbColumnName)
```

Level 2



The **SQLColumnPrivileges** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szTableQualifier</i>	Input	Table qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i> .
UCHAR FAR *	<i>szTableOwner</i>	Input	Owner name. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i> .
UCHAR FAR *	<i>szTableName</i>	Input	Table name.
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .
UCHAR FAR *	<i>szColumnName</i>	Input	String search pattern for column names.
SWORD	<i>cbColumnName</i>	Input	Length of <i>szColumnName</i> .

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE.

## Diagnostics

When **SQLColumnPrivileges** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLColumnPrivileges** and explains each one in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.

(1 of 2)

SQLSTATE	Error	Description
S1090	Invalid string or buffer length	(DM) The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.  The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name. See “Usage.”
S1C00	Driver not capable	A table qualifier was specified, but the driver or data source does not support qualifiers.  A table owner was specified, but the driver or data source does not support owners.  A string search pattern was specified for the column name, but the data source does not support search patterns for that argument.  The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

## Usage

**SQLColumnPrivileges** returns the results as a standard result set, ordered by TABLE\_QUALIFIER, TABLE\_OWNER, TABLE\_NAME, COLUMN\_NAME, GRANTOR, and GRANTEE. The following table lists the columns in the result set.



**Important:** *SQLColumnPrivileges* might not return all columns. For example, the driver might not return information about pseudocolumns, such as Informix ROWID. Applications can use any valid column, regardless of whether it is returned by **SQLColumnPrivileges**.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE\_QUALIFIER, TABLE\_OWNER, TABLE\_NAME, and COLUMN\_NAME columns, an application can call **SQLGetInfo** with the SQL\_MAX\_QUALIFIER\_NAME\_LEN, SQL\_MAX\_OWNER\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	VARCHAR(128)	Table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	VARCHAR(128)	Table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	VARCHAR(128) not NULL	Table identifier.
COLUMN_NAME	VARCHAR(128) not NULL	Column identifier.
GRANTOR	VARCHAR(128)	Identifier of the user who granted the privilege; NULL if not applicable to the data source.

(1 of 2)

Column Name	Data Type	Comments
GRANTEE	VARCHAR(128) not NULL	Identifier of the user to whom the privilege was granted.
PRIVILEGE	VARCHAR(128) not NULL	Identifies the column privilege. Can be one of the following or others supported by the data source when implementation defined:  SELECT: The grantee is permitted to retrieve data for the column.  INSERT: The grantee is permitted to provide data for the column in new rows that are inserted into the associated table.  UPDATE: The grantee is permitted to update data in the column.  REFERENCES: The grantee is permitted to refer to the column within a constraint (for example, a unique, referential, or table-check constraint).
IS_GRANTABLE	VARCHAR(3)	Indicates whether the grantee is permitted to grant the privilege to other users; "YES", "NO", or NULL if unknown or not applicable to the data source.

(2 of 2)

The *szColumnName* argument accepts a search pattern. For more information about valid search patterns, see [“Search Pattern Arguments” on page 13-8](#).

## Code Example

For a code example of a similar function, see [SQLColumns](#).

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning the columns in a table or tables	<b>SQLColumns</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning privileges for a table or tables	<b>SQLTablePrivileges</b>
Returning a list of tables in a data source	<b>SQLTables</b>

## SQLColumns

Level 1

**SQLColumns** returns the list of column names in specified tables. The driver returns this information as a result set on the specified *hstmt*.

### Syntax

```
RETCODE SQLColumns(hstmt, szTableQualifier,
                  cbTableQualifier, szTableOwner, cbTableOwner, szTableName,
                  cbTableName, szColumnName, cbColumnName)
```

The **SQLColumns** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szTableQualifier</i>	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i> .
UCHAR FAR *	<i>szTableOwner</i>	Input	String search pattern for owner names. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i> .
UCHAR FAR *	<i>szTableName</i>	Input	String search pattern for table names.
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .
UCHAR FAR *	<i>szColumnName</i>	Input	String search pattern for column names.
SWORD	<i>cbColumnName</i>	Input	Length of <i>szColumnName</i> .

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLColumns** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLColumns** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.

(1 of 2)



SQLSTATE	Error	Description
S1090	Invalid string or buffer length	<p>(DM) The value of one of the name length arguments was less than 0 but not equal to SQL_NTS.</p> <p>The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name. The maximum length of each qualifier or name can be obtained by calling <b>SQLGetInfo</b> with the <i>InfoType</i> values (see “Usage” on page 13-184).</p>
S1C00	Driver not capable	<p>A table qualifier was specified, but the driver or data source does not support qualifiers.</p> <p>A table owner was specified, but the driver or data source does not support owners.</p> <p>A string search pattern was specified for the table owner, table name, or column name, but the data source does not support search patterns for one or more of those arguments.</p> <p>The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options is not supported by the driver or data source.</p>
S1T00	Time-out expired	<p>The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b>, SQL_QUERY_TIMEOUT.</p>

(2 of 2)

## Usage

This function is typically used before statement execution to retrieve information about columns for a table or tables from the catalog of the data source. In contrast, **SQLColAttributes** and **SQLDescribeCol** describe the columns in a result set, and **SQLNumResultCols** returns the number of columns in a result set.



**Important:** *SQLColumns* might not return all columns. For example, the driver might not return information about pseudocolumns, such as Informix ROWID. Applications can use any valid column, regardless of whether it is returned by *SQLColumns*.

*SQLColumns* returns the results as a standard result set, ordered by TABLE\_QUALIFIER, TABLE\_OWNER, TABLE\_NAME, COLUMN\_NAME, DATA\_TYPE, and S TYPE\_NAME. The following table lists the columns in the result set. Additional columns beyond column 12 (**REMARKS**) can be defined by the driver.

The lengths of VARCHAR columns shown in the following table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE\_QUALIFIER, TABLE\_OWNER, TABLE\_NAME, and COLUMN\_NAME columns, an application can call **SQLGetInfo** with the SQL\_MAX\_QUALIFIER\_NAME\_LEN, SQL\_MAX\_OWNER\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	VARCHAR(128)	Table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	VARCHAR(128)	Table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	VARCHAR(128) not NULL	Table identifier.
COLUMN_NAME	VARCHAR(128) not NULL	Column identifier.

(1 of 3)

Column Name	Data Type	Comments
DATA_TYPE	SMALLINT not NULL	SQL data type. For a list of valid ODBC SQL data types, see <a href="#">Appendix C</a> . For information on how Informix data types map to ODBC SQL data types, see “ <a href="#">Mapping Data Types</a> ” on page 1-15.
TYPE_NAME	VARCHAR(128) not NULL	Data-source-dependent data-type name; for example, CHAR, VARCHAR, MONEY, LONG VARBINARY, or CHAR ( ) for BIT DATA.
PRECISION	INTEGER	The precision of the column on the data source. For precision information, see “ <a href="#">Precision, Scale, Length, and Display Size</a> ” on page C-8.
LENGTH	INTEGER	The length in bytes of data transferred on an <b>SQLGetData</b> or <b>SQLFetch</b> operation if SQL_C_DEFAULT is specified. For numeric data, this size might be different than the size of the data stored on the data source. This value is the same as the PRECISION column for character or binary data. For more information about length, see “ <a href="#">Precision, Scale, Length, and Display Size</a> ” on page C-8.
SCALE		The scale of the column on the data source. For more scale information, see “ <a href="#">Precision, Scale, Length, and Display Size</a> ” on page C-8. NULL is returned for data types where scale is not applicable.

(2 of 3)

Column Name	Data Type	Comments
RADIX	SMALLINT	<p>For numeric data types, either 10 or 2. If it is 10, the values in PRECISION and SCALE give the number of decimal digits allowed for the column. For example, a DECIMAL(12,5) column would return a RADIX of 10, a PRECISION of 12, and a SCALE of 5; a FLOAT column could return a RADIX of 10, a PRECISION of 15, and a SCALE of NULL.</p> <p>If it is 2, the values in PRECISION and SCALE give the number of bits allowed in the column. For example, a FLOAT column could return a RADIX of 2, a PRECISION of 53, and a SCALE of NULL.</p> <p>NULL is returned for data types where RADIX is not applicable.</p>
NULLABLE	SMALLINT not NULL	<p>SQL_NO_NULLS if the column does not accept NULL values.</p> <p>SQL_NULLABLE if the column accepts NULL values.</p> <p>SQL_NULLABLE_UNKNOWN if it is not known if the column accepts NULL values.</p>
REMARKS	VARCHAR(254)	A description of the column.

(3 of 3)

The *szTableOwner*, *szTableName*, and *szColumnName* arguments accept search patterns. For more information about valid search patterns, see [“Search Pattern Arguments”](#) on page 13-8.

## Code Example

In the following example, an application calls **SQLColumns** to return a result set that describes each column in the **EMPLOYEE** table. It then calls **SQLBindCol** to bind the columns in the result set to the storage locations. Finally, the application fetches each row of data with **SQLFetch** and processes it.

```
#define STR_LEN 128+1
#define REM_LEN 254+1

/* Declare storage locations for result set data */

UCHAR  szQualifier[STR_LEN], szOwner[STR_LEN];
UCHAR  szTableName[STR_LEN], szColName[STR_LEN];
UCHAR  szTypeName[STR_LEN], szRemarks[REM_LEN];
SDWORD Precision, Length;
SWORD  DataType, Scale, Radix, Nullable;

/* Declare storage locations for bytes available to return */

SDWORD cbQualifier, cbOwner, cbTableName, cbColName;
SDWORD cbTypeName, cbRemarks, cbDataType, cbPrecision;
SDWORD cbLength, cbScale, cbRadix, cbNullable;

retcode = SQLColumns(hstmt,
                    NULL, 0,          /* All qualifiers */
                    NULL, 0,          /* All owners */
                    "EMPLOYEE", SQL_NTS, /* EMPLOYEE table */
                    NULL, 0);        /* All columns */
if (retcode == SQL_SUCCESS) {

    /* Bind columns in result set to storage locations */

    SQLBindCol(hstmt, 1, SQL_C_CHAR, szQualifier, STR_LEN, &cbQualifier);
    SQLBindCol(hstmt, 2, SQL_C_CHAR, szOwner, STR_LEN, &cbOwner);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, szTableName, STR_LEN, &cbTableName);
    SQLBindCol(hstmt, 4, SQL_C_CHAR, szColName, STR_LEN, &cbColName);
    SQLBindCol(hstmt, 5, SQL_C_SSHORT, &DataType, 0, &cbDataType);
    SQLBindCol(hstmt, 6, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
    SQLBindCol(hstmt, 7, SQL_C_SLONG, &Precision, 0, &cbPrecision);
    SQLBindCol(hstmt, 8, SQL_C_SLONG, &Length, 0, &cbLength);
    SQLBindCol(hstmt, 9, SQL_C_SSHORT, &Scale, 0, &cbScale);
    SQLBindCol(hstmt, 10, SQL_C_SSHORT, &Radix, 0, &cbRadix);
    SQLBindCol(hstmt, 11, SQL_C_SSHORT, &Nullable, 0, &cbNullable);
    SQLBindCol(hstmt, 12, SQL_C_CHAR, szRemarks, REM_LEN, &cbRemarks);

    while(TRUE) {
        retcode = SQLFetch(hstmt);
        if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
            show_error();
        }
        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
```

```

        ...; /* Process fetched data */
    } else {
        break;
    }
}
}

```

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning privileges for a column or columns	<b>SQLColumnPrivileges</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning table statistics and indexes	<b>SQLStatistics</b>
Returning a list of tables in a data source	<b>SQLTables</b>
Returning privileges for a table or tables	<b>SQLTablePrivileges</b>

## SQLConnect

**SQLConnect** loads a driver and establishes a connection to a data source. The connection handle references where all information about the connection, including status, transaction state, and error information is stored.

### Syntax

```
RETCODE SQLConnect(hdbc, szDSN, cbDSN, szUID, cbUID,
  szAuthStr, cbAuthStr)
```

The **SQLConnect** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle
UCHAR FAR *	<i>szDSN</i>	Input	Data-source name
SWORD	<i>cbDSN</i>	Input	Length of <i>szDSN</i>
UCHAR FAR *	<i>szUID</i>	Input	User identifier
SWORD	<i>cbUID</i>	Input	Length of <i>szUID</i>
UCHAR FAR *	<i>szAuthStr</i>	Input	Authentication string (typically the password)
SWORD	<i>cbAuthStr</i>	Input	Length of <i>szAuthStr</i>

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or  
SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLConnect** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLConnect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08001	Unable to connect to data source	The driver could not establish a connection with the data source.
08002	Connection in use	(DM) The specified <i>hdbc</i> had already been used to establish a connection with a data source, and the connection was still open.
08004	Data source rejected establishment of connection	The data source rejected the establishment of the connection for implementation-defined reasons.
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
28000	Invalid authorization specification	The value specified for the argument <i>szUID</i> or the value specified for the argument <i>szAuthStr</i> violated restrictions defined by the data source.
IM002	Data source not found and no default driver specified	(DM) The data-source name specified in the argument <i>szDSN</i> was not found in the <b>odbc.ini</b> file, nor was there a default driver specification.  (DM) The <b>odbc.ini</b> file was not found.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the <b>odbc.ini</b> file was not found or could not be loaded for some other reason.

(1 of 3)



SQLSTATE	Error	Description
IM004	Driver <b>SQLAllocEnv</b> failed	(DM) During <b>SQLConnect</b> , the driver manager called the driver <b>SQLAllocEnv</b> function, and the driver returned an error.
IM005	Driver <b>SQLAllocConnect</b> failed	(DM) During <b>SQLConnect</b> , the driver manager called the driver <b>SQLAllocConnect</b> function, and the driver returned an error.
IM006	Driver <b>SQLSetConnectOption</b> failed	(DM) During <b>SQLConnect</b> , the driver manager called the driver <b>SQLSetConnectOption</b> function, and the driver returned an error (function returns SQL_SUCCESS_WITH_INFO).
IM009	Unable to load translation shared library	The driver did not load the translation shared library that was specified for the data source.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	(DM) The driver manager did not allocate memory required to support execution or completion of the function.  The driver did not allocate memory required to support execution or completion of the function.

(2 of 3)

SQLSTATE	Error	Description
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbDSN</i> was less than 0, but not equal to SQL_NTS.
		(DM) The value specified for argument <i>cbDSN</i> exceeded the maximum length for a data-source name.
		(DM) The value specified for argument <i>cbUID</i> was less than 0 but not equal to SQL_NTS.
		(DM) The value specified for argument <i>cbAuthStr</i> was less than 0 but not equal to SQL_NTS.
S1T00	Time-out expired	The time-out period expired before the connection to the data source completed. The time-out period is set through <b>SQLSetConnectOption</b> , <b>SQL_LOGIN_TIMEOUT</b> .

(3 of 3)

## Usage

The driver manager does not load a driver until the application calls a function (**SQLConnect**, **SQLDriverConnect**, or **SQLBrowseConnect**) to connect to the driver. Until that point, the driver manager works with its own handles and manages connection information. When the application calls a connection function, the driver manager checks if a driver is currently loaded for the specified *hdbc*:

- If a driver is not loaded, the driver manager loads the driver and calls **SQLAllocEnv**, **SQLAllocConnect**, **SQLSetConnectOption** (if the application specified any connection options), and the connection function in the driver. The driver manager returns SQLSTATE IM006 (Driver **SQLSetConnectOption** function failed) and **SQL\_SUCCESS\_WITH\_INFO** for the connection function if the driver returns an error for **SQLSetConnectOption**.
- If the specified driver is already loaded on the *hdbc*, the driver manager calls only the connection function in the driver. In this case, the driver must ensure that all connection options for the *hdbc* maintain their current settings.

The driver then allocates handles and initializes itself.

When the application calls **SQLDisconnect**, the driver manager calls **SQLDisconnect** in the driver. However, it does not unload the driver. This keeps the driver in memory for applications that repeatedly connect to and disconnect from a data source. When the application calls **SQLFreeConnect**, the driver manager calls **SQLFreeConnect** and **SQLFreeEnv** in the driver and then unloads the driver.

An INFORMIX-CLI application can establish more than one connection.

After being loaded by the driver manager, a driver can locate its corresponding data-source specification in the **odbc.ini** file and use driver-specific information from the specification to complete its set of required connection information.

## Code Example

In the following example, an application allocates environment and connection handles. It then connects to the **EmpData** data source with the user ID **JohnS** and the password **Sesame** and processes data. When it has finished processing data, it disconnects from the data source and frees the handles.

```
HENV    henv;
HDBC    hdbc;
HSTMT   hstmt;
RETCODE retcode;

retcode = SQLAllocEnv(&henv);           /* Environment handle */
if (retcode == SQL_SUCCESS) {
    retcode = SQLAllocConnect(henv, &hdbc); /* Connection handle */
    if (retcode == SQL_SUCCESS) {

        /* Set login time-out to 5 seconds. */

        SQLSetConnectOption(hdbc, SQL_LOGIN_TIMEOUT, 5);

        /* Connect to data source */

        retcode = SQLConnect(hdbc, "EmpData", SQL_NTS,
                               "JohnS", SQL_NTS,
                               "Sesame", SQL_NTS);

        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
```

```

        /* Process data after successful connection */

        retcode = SQLAllocStmt(hdbc, &hstmt); /* Statement handle */
        if (retcode == SQL_SUCCESS) {
            ...;
            ...;
            ...;
            SQLFreeStmt(hstmt, SQL_DROP);
        }
        SQLDisconnect(hdbc);
    }
    SQLFreeConnect(hdbc);
}
SQLFreeEnv(henv);
}

```

## Related Functions

For information about	See
Allocating a connection handle	<b>SQLAllocConnect</b>
Allocating a statement handle	<b>SQLAllocStmt</b>
Discovering and enumerating values required to connect to a data source	<b>SQLBrowseConnect</b>
Disconnecting from a data source	<b>SQLDisconnect</b>
Connecting to a data source using a connection string or dialog box	<b>SQLDriverConnect</b>
Returning the setting of a connection option	<b>SQLGetConnectOption</b>
Setting a connection option	<b>SQLSetConnectOption</b>

## SQLDataSources

**SQLDataSources** lists data-source names. This function is implemented solely by the driver manager.

### Syntax

```
RETCODE SQLDataSources(henv, fDirection, szDSN, cbDSNMax,
                      pcbDSN, szDescription, cbDescriptionMax, pcbDescription)
```

The **SQLDataSources** function accepts the following arguments.

Type	Argument	Use	Description
HENV	<i>henv</i>	Input	Environment handle.
UWORD	<i>fDirection</i>	Input	Determines whether the driver manager fetches the next data-source name in the list (SQL_FETCH_NEXT) or whether the search starts from the beginning of the list (SQL_FETCH_FIRST).
UCHAR FAR *	<i>szDSN</i>	Output	Pointer to storage for the data-source name.
SWORD	<i>cbDSNMax</i>	Input	Maximum length of the <i>szDSN</i> buffer; this buffer need not be longer than SQL_MAX_DSN_LENGTH + 1.
SWORD FAR *	<i>pcbDSN</i>	Output	Total number of bytes (excluding the null-termination byte) available to return in <i>szDSN</i> . If the number of bytes available to return is greater than or equal to <i>cbDSNMax</i> , the data-source name in <i>szDSN</i> is truncated to <i>cbDSNMax</i> - 1 bytes.

(1 of 2)

Type	Argument	Use	Description
UCHAR FAR *	<i>szDescription</i>	Output	Pointer to storage for the description of the driver associated with the data source.
SWORD	<i>cbDescriptionMax</i>	Input	Maximum length of the <i>szDescription</i> buffer; this buffer should be at least 255 bytes.
SWORD FAR *	<i>pcbDescription</i>	Output	Total number of bytes (excluding the null-termination byte) available to return in <i>szDescription</i> . If the number of bytes available to return is greater than or equal to <i>cbDescriptionMax</i> , the driver description in <i>szDescription</i> is truncated to <i>cbDescriptionMax</i> - 1 bytes.

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLDataSources** returns either SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDataSources** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO).
01004	Data truncated	(DM) The buffer <i>szDSN</i> was not large enough to return the entire data-source name, so the name was truncated. The argument <i>pcbDSN</i> contains the length of the entire data-source name (function returns SQL_SUCCESS_WITH_INFO).  (DM) The buffer <i>szDescription</i> was not large enough to return the entire driver description, so the description was truncated. The argument <i>pcbDescription</i> contains the length of the untruncated data-source description (function returns SQL_SUCCESS_WITH_INFO).
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	(DM) The driver manager did not allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbDSNMax</i> was less than 0. (DM) The value specified for argument <i>cbDescriptionMax</i> was less than 0.
S1103	Direction option out of range	(DM) The value specified for the argument <i>fDirection</i> was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

## Usage

An application can call **SQLDataSources** multiple times to retrieve all data-source names. The driver manager retrieves this information from the **odbc.ini** file. When no more data-source names remain, the driver manager returns `SQL_NO_DATA_FOUND`. If **SQLDataSources** is called with `SQL_FETCH_NEXT` immediately after it returns `SQL_NO_DATA_FOUND`, it returns the first data-source name.

If `SQL_FETCH_NEXT` is passed to **SQLDataSources** the first time that it is called, it returns the first data-source name.

The driver determines how data-source names are mapped to actual data sources.

## Related Functions

For information about	See
Discovering and listing values required to connect to a data source	<b>SQLBrowseConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Connecting to a data source using a connection string or dialog box	<b>SQLDriverConnect</b>
Returning driver descriptions and attributes	<b>SQLDrivers</b>



## SQLDescribeCol

**SQLDescribeCol** returns the result descriptor (column name, type, precision, scale, and nullability) for one column in the result set; it cannot be used to return information about the bookmark column (column 0).

### Syntax

```
RETCODE SQLDescribeCol(hstmt, icol, szColName, cbColNameMax,  
pcbColName, pfSqlType, pcbColDef, pibScale, pfNullable)
```

The **SQLDescribeCol** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>icol</i>	Input	Column number of result data, ordered sequentially left to right, starting at 1.
UCHAR FAR *	<i>szColName</i>	Output	Pointer to storage for the column name. If the column is unnamed or the column name cannot be determined, the driver returns an empty string.
SWORD	<i>cbColNameMax</i>	Input	Maximum length of the <i>szColName</i> buffer.
SWORD FAR *	<i>pcbColName</i>	Output	Total number of bytes (excluding the null-termination byte) available to return in <i>szColName</i> . If the number of bytes available to return is greater than or equal to <i>cbColNameMax</i> , the column name in <i>szColName</i> is truncated to <i>cbColNameMax</i> - 1 bytes.

(1 of 3)

Type	Argument	Use	Description
SWORD FAR *	<i>ptSqlType</i>	Output	<p>The SQL data type of the column. The SQL data type must be one of the following values:</p> <p>SQL_CHAR  SQL_DATE  SQL_DECIMAL  SQL_DOUBLE  SQL_INTEGER  SQL_LONGVARBINARY  SQL_LONGVARCHAR  SQL_REAL  SQL_SMALLINT  SQL_TIMESTAMP  SQL_VARCHAR</p> <p>If the data type cannot be determined, the driver returns 0. For more information, see <a href="#">Appendix C</a>. For information on how Informix data types map to ODBC SQL data types, see <a href="#">“Mapping Data Types” on page 1-15</a></p>

(2 of 3)

Type	Argument	Use	Description
UDWORD FAR *	<i>pcbColDef</i>	Output	The precision of the column on the data source. If the precision cannot be determined, the driver returns 0. For more information on precision, see <a href="#">“Precision, Scale, Length, and Display Size”</a> on page C-8.
SWORD FAR *	<i>pibScale</i>	Output	The scale of the column on the data source. If the scale cannot be determined or is not applicable, the driver returns 0. For more information on scale, see <a href="#">“Precision, Scale, Length, and Display Size”</a> on page C-8.
SWORD FAR *	<i>pfNullable</i>	Output	Indicates whether the column allows one of the following values:  SQL_NO_NULLS: The column does not allow NULL values.  SQL_NULLABLE: The column allows NULL values.  SQL_NULLABLE_UNKNOWN: The driver cannot determine if the column allows NULL values.

(3 of 3)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLDescribeCol** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLDescribeCol** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The buffer <i>szColName</i> was not large enough to return the entire column name, so the column name was truncated. The argument <i>pcbColName</i> contains the length of the untruncated column name (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
24000	Invalid cursor state	The statement associated with the <i>hstmt</i> did not return a result set. There were no columns to describe.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.
S1002	Invalid column number	(DM) The value specified for the argument <i>icol</i> was 0.  The value specified for the argument <i>icol</i> was greater than the number of columns in the result set.

(1 of 2)

SQLSTATE	Error	Description
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multithreaded application.
S1010	Function-sequence error	(DM) The function was called prior to calling <b>SQLPrepare</b> or <b>SQLExecDirect</b> for the <i>hstmt</i> .  (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbColNameMax</i> was less than 0.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

**SQLDescribeCol** can return any SQLSTATE that **SQLPrepare** or **SQLExecute** returns when **SQLDescribeCol** is called after **SQLPrepare** and before **SQLExecute**, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

## Usage

An application typically calls **SQLDescribeCol** after a call to **SQLPrepare** and before or after the associated call to **SQLExecute**. An application can also call **SQLDescribeCol** after a call to **SQLExecDirect**.

**SQLDescribeCol** retrieves the column name, type, and length generated by a SELECT statement. If the column is an expression, *szColName* is either an empty string or a driver-defined name.

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning information about a column in a result set	<b>SQLColAttributes</b>
Fetching a row of data	<b>SQLFetch</b>
Returning the number of result-set columns	<b>SQLNumResultCols</b>
Preparing a statement for execution	<b>SQLPrepare</b>

## SQLDisconnect

**SQLDisconnect** closes the connection associated with a specific connection handle.

### Syntax

```
RETCODE SQLDisconnect(hdbc)
```

The **SQLDisconnect** function accepts the following argument.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

Core

## Diagnostics

When **SQLDisconnect** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLDisconnect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01002	Disconnect error	An error occurred during the disconnect. However, the disconnect succeeded (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
08003	Connection not open	(DM) The connection specified in the argument <i>hdbc</i> was not open.
25000	Invalid transaction state	A transaction was in process on the connection specified by the argument <i>hdbc</i> . The transaction remains active.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> associated with the <i>hdbc</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.

## Usage

If an application calls **SQLDisconnect** after **SQLBrowseConnect** returns `SQL_NEED_DATA` and before it returns a different return code, the driver cancels the connection-browsing process and returns the *hdbc* to an unconnected state.

If an application calls **SQLDisconnect** while an incomplete transaction is associated with the connection handle, the driver returns `SQLSTATE 25000` (Invalid transaction state), indicating that the transaction is unchanged and the connection is open. An incomplete transaction is one that has not been committed or rolled back with **SQLTransact**.

If an application calls **SQLDisconnect** before it has freed every *hstmt* associated with the connection, the driver frees each remaining *hstmt* after it successfully disconnects from the data source.

## Code Example

See **SQLBrowseConnect** and **SQLConnect**.

## Related Functions

For information about	See
Allocating a connection handle	<b>SQLAllocConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Connecting to a data source using a connection string or dialog box	<b>SQLDriverConnect</b>
Freeing a connection handle	<b>SQLFreeConnect</b>
Executing a commit or rollback operation	<b>SQLTransact</b>



## SQLDriverConnect

**SQLDriverConnect** is an alternative to **SQLConnect**. It supports data sources that require more connection information than the three arguments in **SQLConnect**, dialog boxes to prompt the user for all connection information, and data sources that are not defined in the **odbc.ini** file.

**SQLDriverConnect** provides the following connection options:

- You can establish a connection using a connection string that contains the data-source name, one or more user IDs, one or more passwords, and other information required by the data source.
- You can establish a connection using a partial connection string or no additional information; in this case, the driver manager and the driver can each prompt the user for connection information.

Once a connection is established, **SQLDriverConnect** returns the completed connection string. The application can use this string for subsequent connection requests.

### Syntax

```
RETCODE SQLDriverConnect(hdbc, hwnd, szConnStrIn,  
cbConnStrIn, szConnStrOut, cbConnStrOutMax, pcbConnStrOut,  
fDriverCompletion)
```

The **SQLDriverConnect** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
HWND	<i>hwnd</i>	Input	Window handle (platform dependent): For Windows, this is the parent window handle. For UNIX, this is the parent Widget handle. If the window handle is not applicable, or a null pointer is passed, <b>SQLDriverConnect</b> does not present any dialog boxes.
UCHAR FAR *	<i>szConnStrIn</i>	Input	A full connection string (see <a href="#">“Connection Strings” on page 13-99</a> ), a partial connection string, or an empty string.
SWORD	<i>cbConnStrIn</i>	Input	Length of <i>szConnStrIn</i> .
UCHAR FAR	<i>*szConnStrOut</i>	Output	Pointer to storage for the completed connection string. Upon successful connection to the target data source, this buffer contains the completed connection string. Applications should allocate at least 255 bytes for this buffer.
SWORD	<i>cbConnStrOutMax</i>	Input	Maximum length of the <i>szConnStrOut</i> buffer.
SWORD FAR *	<i>pcbConnStrOut</i>	Output	Pointer to the total number of bytes (excluding the null termination byte) available to return in <i>szConnStrOut</i> . If the number of bytes available to return is greater than or equal to <i>cbConnStrOutMax</i> , the completed connection string in <i>szConnStrOut</i> is truncated to <i>cbConnStrOutMax</i> - 1 bytes.
UWORD	<i>fDriverCompletion</i>	Input	Flag that indicates whether driver manager or driver must prompt for more connection information:  SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, or SQL_DRIVER_NOPROMPT.  For more information, see <a href="#">“Driver Manager Guidelines” on page 13-100</a> and <a href="#">“Driver Guidelines” on page 13-101</a> .

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLDriverConnect** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLDriverConnect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The buffer <i>szConnStrOut</i> was not large enough to return the entire connection string, so the connection string was truncated. The argument <i>pcbConnStrOut</i> contains the length of the untruncated connection string (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
01S00	Invalid connection string attribute	An invalid attribute keyword was specified in the connection string ( <i>szConnStrIn</i> ), but the driver connected to the data source anyway (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
08001	Unable to connect to data source	The driver did not establish a connection with the data source.
08002	Connection in use	(DM) The specified <i>hdbc</i> was used already to establish a connection with a data source, and the connection was still open.
08004	Data source rejected establishment of connection	The data source rejected the establishment of the connection for implementation-defined reasons.

(1 of 4)

SQLSTATE	Error	Description
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
28000	Invalid authorization specification	Either the user identifier or the authorization string or both as specified in the connection string ( <i>szConnStrIn</i> ) violated restrictions defined by the data source.
IM002	Data source not found and no default driver specified	(DM) The data-source name specified in the connection string ( <i>szConnStrIn</i> ) was not found in the <b>odbc.ini</b> file, and no default driver specification existed.  (DM) The <b>odbc.ini</b> file could not be found.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data-source specification in the <b>odbc.ini</b> file, or specified by the DRIVER keyword, was not found or was not loaded for some other reason.
IM004	Driver <b>SQLAllocEnv</b> failed	(DM) During <b>SQLDriverConnect</b> , the driver manager called the driver <b>SQLAllocEnv</b> function, and the driver returned an error.
IM005	Driver <b>SQLAllocConnect</b> failed	(DM) During <b>SQLDriverConnect</b> , the driver manager called the driver <b>SQLAllocConnect</b> function, and the driver returned an error.
IM006	Driver <b>SQLSetConnectOption</b> failed	(DM) During <b>SQLDriverConnect</b> , the driver manager called the driver <b>SQLSetConnectOption</b> function, and the driver returned an error.
IM007	No data source or driver specified; dialog prohibited	<i>C fDriverCompletion</i> was SQL_DRIVER_NOPROMPT.
IM008	Dialog failed	(DM) The driver manager attempted to display the SQL Data Sources dialog box but failed.  The driver attempted to display its login dialog box but failed.

(2 of 4)

SQLSTATE	Error	Description
IM009	Unable to load translation shared library	The driver did not load the translation shared library that was specified for the data source or for the connection.
IM010	Data-source name too long	(DM) The attribute value for the DSN keyword was longer than SQL_MAX_DSN_LENGTH characters.
IM011	Driver name too long	(DM) The attribute value for the DRIVER keyword was longer than 255 characters.
IM012	DRIVER keyword syntax error	(DM) The keyword-value pair for the DRIVER keyword contained a syntax error.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver manager did not allocate memory required to support execution or completion of the function.  The driver did not allocate memory required to support execution or completion of the function.

(3 of 4)

SQLSTATE	Error	Description
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbConnStrIn</i> was less than 0 and was not equal to SQL_NTS.  (DM) The value specified for argument <i>cbConnStrOutMax</i> was less than 0.
S1110	Invalid driver completion	(DM) The value specified for the argument <i>fDriverCompletion</i> was not equal to SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, or SQL_DRIVER_NOPROMPT.
S1T00	Time-out expired	The time-out period expired before the connection to the data source completed. The time-out period is set through <b>SQLSetConnectOption</b> , SQL_LOGIN_TIMEOUT.

(4 of 4)

## Usage

**SQLDriverConnect** uses a connection string to specify the information needed to connect to a driver and data source. The connection string is a more flexible interface than the data-source name, user ID, and password used by **SQLConnect**. The application can use the connection string for multiple levels of login authorization or to convey other data-source-specific connection information.

## Connection Strings

A connection string has the following syntax:

```

connection-string ::= empty-string[:] | attribute[:] |
attribute; connection-string
empty-string ::=
attribute ::= attribute-keyword=attribute-value |
DRIVER={attribute-value}
(The braces ({} ) are literal; the application must specify
them.)
attribute-keyword ::= DSN | UID | PWD
| driver-defined-attribute-keyword
attribute-value ::= character-string
driver-defined-attribute-keyword ::= identifier

```

In this example, *character-string* has zero or more characters; *identifier* has one or more characters; *attribute-keyword* is case insensitive; *attribute-value* might be case sensitive; and the value of the DSN keyword does not consist solely of blanks. Because of connection-string and initialization-file grammar, avoid keywords and attribute values that contain the characters [ ] { } ( ) , ; ? \* = ! @ \ . Because of the registry grammar, keywords and data-source names cannot contain the backslash (\).

The connection string might include several driver-defined keywords. Because the DRIVER keyword does not use information from the **odbc.ini** file, the driver must define enough keywords so that a driver can connect to a data source using only the information in the connection string. (For more information, see [“Driver Guidelines” on page 13-101.](#)) The driver defines which keywords are required in order to connect to the data source. For more information about keywords that can be used in a connection string for the INFORMIX-CLI driver, see [“Using a Connection-String to Connect to a Data Source” on page 2-13](#) for UNIX and [“Using a Connection-String to Connect to a Data Source” on page 2-28](#) for Windows.

If any keywords are repeated in the connection string, the driver uses the value associated with the first occurrence of the keyword. If the DSN and DRIVER keywords are included in the same connection string, the driver manager and the driver use the keyword that appears first. The following table describes the attribute values of the DSN, DRIVER, UID, and PWD keywords.

Keyword	Attribute value description
DSN	Name of a data source as returned by <b>SQLDataSources</b> or the data sources dialog box of <b>SQLDriverConnect</b>
DRIVER	Description of the driver as returned by the <b>SQLDrivers</b> function
UID	A user ID
PWD	The password that corresponds to the user ID, or an empty string if no password exists for the user ID (PWD=;)

### ***Driver Manager Guidelines***

The driver manager constructs a connection string to pass to the driver in the *szConnStrIn* argument of the driver **SQLDriverConnect** function. The driver manager does not modify the *szConnStrIn* argument passed to it by the application.

If the connection string specified by the application contains the DSN keyword or does not contain either the DSN or DRIVER keywords, the action of the driver manager is based on the value of the *fDriverCompletion* argument, as follows:

- **SQL\_DRIVER\_PROMPT**  
The driver manager displays the Data Sources dialog box. It constructs a connection string from the data source name returned by the dialog box and any other keywords passed to it by the application. If the data-source name returned by the dialog box is empty, the driver manager specifies the keyword-value pair DSN=Default.
- **SQL\_DRIVER\_COMPLETE** or **SQL\_DRIVER\_COMPLETE\_REQUIRED**  
If the connection string specified by the application includes the DSN keyword, the driver manager copies the connection string specified by the application. Otherwise, it takes the same actions as it does when *fDriverCompletion* is **SQL\_DRIVER\_PROMPT**.
- **SQL\_DRIVER\_NOPROMPT**  
The driver manager copies the connection string specified by the application.



If the connection string specified by the application contains the DRIVER keyword, the driver manager copies the connection string specified by the application.

Using the connection string that it constructed, the driver manager determines which driver to use, loads that driver, and passes the connection string that it has constructed to the driver. For more information about the interaction of the driver manager and the driver, see “Usage” on page 13-78. If the connection string contains the DSN keyword or does not contain either the DSN or the DRIVER keyword, the driver manager determines which driver to use, as follows:

1. If the connection string contains the DSN keyword, the driver manager retrieves the driver associated with the data source from the **odbc.ini** file.
2. If the connection string does not contain the DSN keyword or the data source is not found, the driver manager retrieves the driver associated with the default data source from the **odbc.ini** file. However, the driver manager does not change the value of the DSN keyword in the connection string.
3. If the data source is not found and the Default data source is not found, the driver manager returns SQL\_ERROR with SQLSTATE IM002 (Data source not found and no default driver specified).

### ***Driver Guidelines***

The driver checks if the connection string passed to it by the driver manager contains the DSN or DRIVER keyword. If the connection string contains the DRIVER keyword, the driver cannot retrieve information about the data source from the **odbc.ini** file. If the connection string contains the DSN keyword or does not contain either the DSN or the DRIVER keyword, the driver can retrieve information about the data source from the **odbc.ini** file, as follows:

1. If the connection string contains the DSN keyword, the driver retrieves the information for the specified data source.
2. If the connection string does not contain the DSN keyword or the specified data source is not found, the driver retrieves the information for the default data source.

The driver uses any information that it retrieves from the **odbc.ini** file to augment the information passed to it in the connection string. If the information in the **odbc.ini** file duplicates information in the connection string, the driver uses the information in the connection string.

Based on the value of *fDriverCompletion*, the driver prompts the user for connection information, such as the user ID and password, and connects to the data source, as follows:

- **SQL\_DRIVER\_PROMPT**

The driver displays a dialog box, using the values from the connection string and **odbc.ini** file as initial values. When the user exits the dialog box, the driver connects to the data source. It also constructs a connection string from the value of the DSN or DRIVER keyword in *szConnStrIn* and the information returned from the dialog box. It places this connection string in the buffer referenced by *szConnStrOut*.

- **SQL\_DRIVER\_COMPLETE** or **SQL\_DRIVER\_COMPLETE\_REQUIRED**

If the connection string contains enough correct information, the driver connects to the data source and copies *szConnStrIn* to *szConnStrOut*. If any information is missing or incorrect, the driver takes the same actions as it does when *fDriverCompletion* is **SQL\_DRIVER\_PROMPT**, except that if *fDriverCompletion* is **SQL\_DRIVER\_COMPLETE\_REQUIRED**, the driver disables the controls for any information that is not required to connect to the data source.

- **SQL\_DRIVER\_NOPROMPT**

If the connection string contains enough information, the driver connects to the data source and copies *szConnStrIn* to *szConnStrOut*. Otherwise, the driver returns **SQL\_ERROR** for **SQLDriverConnect**.

After successfully connecting to the data source, the driver also sets *pcbConnStrOut* to the length of *szConnStrOut*.

If the user cancels a dialog box presented by the driver manager or the driver, **SQLDriverConnect** returns **SQL\_NO\_DATA\_FOUND**.

For information about how the driver manager and the driver interact during the connection process, see “Usage” on page 13-78.

## Connection Options

The `SQL_LOGIN_TIMEOUT` connection option, set using **SQLSetConnectOption**, defines the number of seconds to wait for a login request to complete before returning to the application. If the user is prompted to complete the connection string, a waiting period for each login request begins after the user has dismissed each dialog box.

The driver opens the connection in `SQL_MODE_READ_WRITE` access mode by default. To set the access mode to `SQL_MODE_READ_ONLY`, the application must call **SQLSetConnectOption** with the `SQL_ACCESS_MODE` option before calling **SQLDriverConnect**.

If a default translation shared library is specified in the `odbc.ini` file for the data source, the driver loads it. A different translation shared library can be loaded by calling **SQLSetConnectOption** with the `SQL_TRANSLATE_DLL` option. A translation option can be specified by calling **SQLSetConnectOption** with the `SQL_TRANSLATE_OPTION` option.

## Related Functions

For information about	See
Allocating a connection handle	<b>SQLAllocConnect</b>
Discovering and enumerating values required to connect to a data source	<b>SQLBrowseConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Disconnecting from a data source	<b>SQLDisconnect</b>
Returning driver descriptions and attributes	<b>SQLDrivers</b>
Freeing a connection handle	<b>SQLFreeConnect</b>
Setting a connection option	<b>SQLSetConnectOption</b>

## Level 2

## SQLDrivers

**SQLDrivers** lists driver descriptions and driver-attribute keywords. This function is implemented solely by the driver manager. **SQLDrivers** is an ODBC 2.0 function.

### Syntax

```
RETCODE SQLDrivers(henv, fDirection, szDriverDesc,
                  cbDriverDescMax, pcbDriverDesc, szDriverAttributes,
                  cbDrvAttrMax, pcbDrvAttr)
```

The **SQLDrivers** function accepts the following arguments.

Type	Argument	Use	Description
HENV	<i>henv</i>	Input	Environment handle.
UWORD	<i>fDirection</i>	Input	Determines whether the driver manager fetches the next driver description in the list (SQL_FETCH_NEXT) or whether the search starts from the beginning of the list (SQL_FETCH_FIRST).
UCHAR FAR *	<i>szDriverDesc</i>	Output	Pointer to storage for the driver description.
SWORD	<i>cbDriverDescMax</i>	Input	Maximum length of the <i>szDriverDesc</i> buffer.
SWORD FAR *	<i>pcbDriverDesc</i>	Output	Total number of bytes (excluding the null-termination byte) available to return in <i>szDriverDesc</i> . If the number of bytes available to return is greater than or equal to <i>cbDriverDescMax</i> , the driver description in <i>szDriverDesc</i> is truncated to <i>cbDriverDescMax</i> - 1 bytes.

(1 of 2)

Type	Argument	Use	Description
UCHAR FAR *	<i>szDriverAttributes</i>	Output	Pointer to storage for the list of driver attribute value pairs (see “Usage” on page 13-107).
SWORD	<i>cbDrvrAttrMax</i>	Input	Maximum length of the <i>szDriverAttributes</i> buffer.
SWORD FAR *	<i>pcbDrvrAttr</i>	Output	Total number of bytes (excluding the null-termination byte) available to return in <i>szDriverAttributes</i> . If the number of bytes available to return is greater than or equal to <i>cbDrvrAttrMax</i> , the list of attribute-value pairs in <i>szDriverAttributes</i> is truncated to <i>cbDrvrAttrMax</i> – 1 bytes.

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLDrivers** returns either SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDrivers** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	(DM) The buffer <i>szDriverDesc</i> was not large enough to return the entire driver description, so the description was truncated. The argument <i>pcbDriverDesc</i> contains the length of the entire driver description (function returns SQL_SUCCESS_WITH_INFO).  (DM) The buffer <i>szDriverAttributes</i> was not large enough to return the entire list of attribute-value pairs, so the list was truncated. The argument <i>pcbDrvAttr</i> contains the length of the untruncated list of attribute-value pairs (function returns SQL_SUCCESS_WITH_INFO).
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	(DM) The driver manager did not allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbDriverDescMax</i> was less than 0.  (DM) The value specified for argument <i>cbDrvAttrMax</i> was less than 0 or equal to 1.
S1103	Direction option out of range	(DM) The value specified for the argument <i>fDirection</i> was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

## Usage

**SQLDrivers** returns the driver description in the *szDriverDesc* argument. It returns additional information about the driver in the *szDriverAttributes* argument as a list of keyword-value pairs. Each pair is terminated with a null byte, and the entire list is terminated with a null byte (that is, 2 null bytes mark the end of the list).

If *szDriverAttributes* cannot hold the entire list, the list is truncated, **SQLDrivers** returns SQLSTATE 01004 (Data truncated), and the length of the list (excluding the final null-termination byte) is returned in *pcbDrvrAttr*.

Driver attribute keywords are added during the installation procedure. For more information, see [“Understanding the odbc.ini File” on page 1-7](#).

An application can call **SQLDrivers** multiple times to retrieve all driver descriptions. The driver manager retrieves this information from the **odbcinst.ini** file. When no more driver descriptions remain, **SQLDrivers** returns SQL\_NO\_DATA\_FOUND. If **SQLDrivers** is called with SQL\_FETCH\_NEXT immediately after it returns SQL\_NO\_DATA\_FOUND, it returns the first driver description.

If SQL\_FETCH\_NEXT passes to **SQLDrivers** the first time it is called, **SQLDrivers** returns the first data-source name.

Because **SQLDrivers** is implemented in the driver manager, it is supported for all drivers, regardless of the conformance level of a particular driver.

## Related Functions

For information about	See
Discovering and listing values required to connect to a data source	<b>SQLBrowseConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Returning data source names	<b>SQLDataSources</b>
Connecting to a data source using a connection string or dialog box	<b>SQLDriverConnect</b>

## SQLError

**SQLError** returns error or status information.

### Syntax

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

The **SQLError** function accepts the following arguments.

Type	Argument	Use	Description
HENV	<i>henv</i>	Input	Environment handle or SQL_NULL_HENV.
HDBC	<i>hdbc</i>	Input	Connection handle or SQL_NULL_HDBC.
HSTMT	<i>hstmt</i>	Input	Statement handle or SQL_NULL_HSTMT.
UCHAR FAR *	<i>szSqlState</i>	Output	SQLSTATE as null-terminated string. For a list of SQLSTATE values, see <a href="#">Appendix A “INFORMIX-CLI Error Codes.”</a>
SDWORD FAR *	<i>pfNativeError</i>	Output	Native error code (specific to the data source).
UCHAR FAR *	<i>szErrorMsg</i>	Output	Pointer to storage for the error message text.
SWORD	<i>cbErrorMsgMax</i>	Input	Maximum length of the <i>szErrorMsg</i> buffer. This value must be less than or equal to SQL_MAX_MESSAGE_LENGTH - 1.
SWORD FAR *	<i>pcbErrorMsg</i>	Output	Pointer to the total number of bytes (excluding the null-termination byte) available to return in <i>szErrorMsg</i> . If the number of bytes available to return is greater than or equal to <i>cbErrorMsgMax</i> , the error message text in <i>szErrorMsg</i> is truncated to <i>cbErrorMsgMax</i> - 1 bytes.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_ERROR, or SQL\_INVALID\_HANDLE



## Diagnostics

**SQLError** does not post error values for itself. **SQLError** returns `SQL_NO_DATA_FOUND` when it cannot retrieve any error information (in which case `szSqlState` equals `00000`). If **SQLError** cannot access error values for any reason that would normally return `SQL_ERROR`, **SQLError** returns `SQL_ERROR` but does not post any error values. If the buffer for the error message is too short, **SQLError** returns `SQL_SUCCESS_WITH_INFO` but still does not return a `SQLSTATE` value for **SQLError**.

To determine that a truncation occurred in the error message, an application can compare `cbErrorMsgMax` to the actual length of the message text written to `pcbErrorMsg`.

## Usage

An application typically calls **SQLError** when a previous call to an INFORMIX-CLI function returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`. However, any INFORMIX-CLI function can post zero or more errors each time that it is called, so an application can call **SQLError** after any INFORMIX-CLI function call.

**SQLError** retrieves an error from the data structure associated with the rightmost non-null handle argument. An application requests error information, as follows:

- To retrieve errors associated with an environment, the application passes the corresponding `henv` and includes `SQL_NULL_HDBC` and `SQL_NULL_HSTMT` in `hdbc` and `hstmt`, respectively. The driver returns the error status of the INFORMIX-CLI function most recently called with the same `henv`.
- To retrieve errors associated with a connection, the application passes the corresponding `hdbc` plus an `hstmt` equal to `SQL_NULL_HSTMT`. In such a case, the driver ignores the `henv` argument. The driver returns the error status of the INFORMIX-CLI function most recently called with the `hdbc`.

- To retrieve errors associated with a statement, an application passes the corresponding *hstmt*. If the call to **SQLError** contains a valid *hstmt*, the driver ignores the *hdbc* and *henv* arguments. The driver returns the error status of the INFORMIX-CLI function most recently called with the *hstmt*.
- To retrieve multiple errors for a function call, an application calls **SQLError** multiple times. For each error, the driver returns `SQL_SUCCESS` and removes that error from the list of available errors.

When no additional information for the rightmost non-null handle remains, **SQLError** returns `SQL_NO_DATA_FOUND`. In this case, *szSqlState* equals `00000` (Success), *pfNativeError* is undefined, *pcbErrorMsg* equals `0`, and *szErrorMsg* contains a single null-termination byte (unless *cbErrorMsgMax* equals `0`).

The driver manager stores error information in its *henv*, *hdbc*, and *hstmt* structures. Similarly, the driver stores error information in its *henv*, *hdbc*, and *hstmt* structures. When the application calls **SQLError**, the driver manager checks if any errors exist in its structure for the specified handle. If errors exist for the specified handle, it returns the first error; if no errors exist, it calls **SQLError** in the driver.

The driver manager can store up to 64 errors with an *henv* and its associated *hdbcs* and *hstmts*. When this limit is reached, the driver manager discards any subsequent errors posted on the *henv*, *hdbcs*, or *hstmts* of the driver manager. The number of errors that a driver can store is driver dependent.

An error is removed from the structure associated with a handle when **SQLError** is called for that handle and returns the error. All errors stored for a specific handle are removed when the handle is used in a subsequent function call. For example, errors on an *hstmt* that were returned by **SQLExecDirect** are removed when **SQLExecDirect** or **SQLTables** is called with that *hstmt*. The errors stored on a specific handle are not removed as the result of a call to a function using an associated handle of a different type. For example, errors on an *hdbc* that were returned by **SQLNativeSql** are not removed when **SQLError** or **SQLExecDirect** is called with an *hstmt* associated with that *hdbc*.

For more information about error codes, see [Appendix A](#).

## Related Functions

None.

---

## SQLExecDirect

**SQLExecDirect** executes a preparable statement, using the current values of the parameter-marker variables if any parameters exist in the statement. **SQLExecDirect** is the fastest way to submit an SQL statement for one-time execution.

## Syntax

```
RETCODE SQLExecDirect(hstmt, szSqlStr, cbSqlStr)
```

The **SQLExecDirect** function uses the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
UCHAR FAR *	<i>szSqlStr</i>	Input	SQL statement to be executed
SDWORD	<i>cbSqlStr</i>	Input	Length of <i>szSqlStr</i>

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NEED\_DATA, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLExecDirect** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLExecDirect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	<p>The argument <i>szSqlStr</i> contained an SQL statement that contained a character or binary parameter or literal, and the value exceeded the maximum length of the associated table column.</p> <p>The argument <i>szSqlStr</i> contained an SQL statement that contained a numeric parameter or literal, and the fractional part of the value was truncated.</p> <p>The argument <i>szSqlStr</i> contained an SQL statement that contained a date or time parameter or literal, and a time-stamp value was truncated.</p>
01006	Privilege not revoked	The argument <i>szSqlStr</i> contained a REVOKE statement, and the user did not have the specified privilege (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
01S03	No rows updated or deleted	The argument <i>szSqlStr</i> contained a positioned UPDATE or DELETE statement, and no rows were updated or deleted (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).

(1 of 6)

SQLSTATE	Error	Description
01S04	More than one row updated or deleted	The argument <i>szSqlStr</i> contained a positioned UPDATE or DELETE statement, and more than one row was updated or deleted (function returns SQL_SUCCESS_WITH_INFO).
07001	Wrong number of parameters	The number of parameters specified in <b>SQLBindParameter</b> was less than the number of parameters in the SQL statement contained in the argument <i>szSqlStr</i> .
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
21S01	Insert value list does not match column list.	The argument <i>szSqlStr</i> contained an INSERT statement, and the number of values to be inserted did not match the degree of the derived table.
21S02	Degree of derived table does not match column list.	The argument <i>szSqlStr</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.
22003	Numeric value out of range	The argument <i>szSqlStr</i> contained an SQL statement that contained a numeric parameter or literal, and the value caused the whole (as opposed to fractional) part of the number to be truncated when assigned to the associated table column.
22005	Error in assignment	The argument <i>szSqlStr</i> contained an SQL statement that contained a parameter or literal, and the value was incompatible with the data type of the associated table column.
22008	Datetime field overflow	The argument <i>szSqlStr</i> contained an SQL statement that contained a date, time, or time-stamp parameter or literal, and the value was, respectively, an invalid date, time, or time stamp.

(2 of 6)

SQLSTATE	Error	Description
22012	Division by zero	The argument <i>szSqlStr</i> contained an SQL statement that contained an arithmetic expression that caused division by zero.
23000	Integrity-constraint violation	The argument <i>szSqlStr</i> contained an SQL statement that contained a parameter or literal. The parameter value was NULL for a column defined as NOT NULL in the associated table column, a duplicate value was supplied for a column constrained to contain only unique values, or some other integrity constraint was violated.
24000	Invalid cursor state	A cursor was already opened on the statement handle.  The argument <i>szSqlStr</i> contained a positioned UPDATE or DELETE statement, but the cursor was positioned before the start of the result set or after the end of the result set.
34000	Invalid cursor name	The argument <i>szSqlStr</i> contained a positioned UPDATE or DELETE statement, but the cursor referenced by the statement being executed was not open.
37000	Syntax error or access violation	The argument <i>szSqlStr</i> contained an SQL statement that was not preparable or contained a syntax error.
40001	Serialization failure	The transaction to which the SQL statement contained in the argument <i>szSqlStr</i> belonged was terminated to prevent deadlock.
42000	Syntax error or access violation	The user did not have permission to execute the SQL statement contained in the argument <i>szSqlStr</i> .
S0001	Base table or view already exists.	The argument <i>szSqlStr</i> contained a CREATE TABLE or CREATE VIEW statement, but the table name or view name specified already exists.

(3 of 6)

SQLSTATE	Error	Description
S0002	Table or view not found	<p>The argument <i>szSqlStr</i> contained a DROP TABLE or a DROP VIEW statement, but the specified table name or view name did not exist.</p> <p>The argument <i>szSqlStr</i> contained an ALTER TABLE statement, but the specified table name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE VIEW statement, but a table name or view name defined by the query specification did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE INDEX statement, but the specified table name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a GRANT or REVOKE statement, but the specified table name or view name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a SELECT statement, but a specified table name or view name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a DELETE, INSERT, or UPDATE statement, but the specified table name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE TABLE statement, but a table specified in a constraint (referencing a table other than the one being created) did not exist.</p>
S0011	Index already exists.	The argument <i>szSqlStr</i> contained a CREATE INDEX statement, but the specified index name already existed.
S0012	Index not found	The argument <i>szSqlStr</i> contained a DROP INDEX statement, but the specified index name did not exist.

(4 of 6)

SQLSTATE	Error	Description
S0021	Column already exists.	The argument <i>szSqlStr</i> contained an ALTER TABLE statement, but the column specified in the ADD clause is not unique or identifies an existing column in the base table.
S0022	Column not found	<p>The argument <i>szSqlStr</i> contained a CREATE INDEX statement, but one or more of the column names specified in the column list did not exist.</p> <p>The argument <i>szSqlStr</i> contained a GRANT or REVOKE statement, but a specified column name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a SELECT, DELETE, INSERT, or UPDATE statement, but a specified column name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE TABLE statement, but a column specified in a constraint (referencing a table other than the one being created) did not exist.</p>
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multithreaded application.
S1009	Invalid argument value	(DM) The argument <i>szSqlStr</i> was a null pointer.

(5 of 6)



SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The argument <i>cbSqlStr</i> was less than or equal to 0, but not equal to SQL_NTS.  A parameter value, set with <b>SQLBindParameter</b> , was a null pointer and the parameter length value was not 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.  A parameter value, set with <b>SQLBindParameter</b> , was not a null pointer and the parameter length value was less than 0, but was not SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.
S1109	Invalid cursor position	The argument <i>szSqlStr</i> contained a positioned UPDATE or DELETE statement, but the cursor was positioned (by <b>SQLExtendedFetch</b> ) on a row for which the value in the <i>rgfRowStatus</i> array in <b>SQLExtendedFetch</b> was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(6 of 6)

## Usage

The application calls **SQLExecDirect** to send an SQL statement to the data source. The driver modifies the statement to use the form of SQL used by the data source then submits it to the data source. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL. For a description of SQL statement grammar, see [Appendix B](#).

The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL statement at the appropriate position.

If the SQL statement is a SELECT statement, and if the application called **SQLSetCursorName** to associate a cursor with an *hstmt*, the driver uses the specified cursor. Otherwise, the driver generates a cursor name.

If the data source is in manual-commit mode (requiring explicit transaction initiation), and a transaction has not been initiated, the driver initiates a transaction before it sends the SQL statement.

If an application uses **SQLExecDirect** to submit a COMMIT or ROLLBACK statement, it is not interoperable between DBMS products. To commit or roll back a transaction, call **SQLTransact**.

If **SQLExecDirect** encounters a data-at-execution parameter, it returns SQL\_NEED\_DATA. The application sends the data using **SQLParamData** and **SQLPutData**. For more information, see **SQLBindParameter**, **SQLParamData**, **SQLParamOptions**, and **SQLPutData**.

## Code Example

See **SQLBindCol**, **SQLExtendedFetch**, **SQLGetData**, and **SQLProcedures**.

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Executing a prepared SQL statement	<b>SQLExecute</b>

---

<b>For information about</b>	<b>See</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning a cursor name	<b>SQLGetCursorName</b>
Fetching part or all of a column of data	<b>SQLGetData</b>
Returning the next parameter to send data for	<b>SQLParamData</b>
Preparing a statement for execution	<b>SQLPrepare</b>
Sending parameter data at execution time	<b>SQLPutData</b>
Setting a cursor name	<b>SQLSetCursorName</b>
Setting a statement option	<b>SQLSetStmtOption</b>
Executing a commit or rollback operation	<b>SQLTransact</b>

---

## SQLExecute

**SQLExecute** executes a prepared statement, using the current values of the parameter-marker variables if any parameter markers exist in the statement.

### Syntax

```
RETCODE SQLExecute(hstmt)
```

The **SQLExecute** statement accepts the following argument.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NEED\_DATA, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

### Diagnostics

When **SQLExecute** returns either SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLExecute** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The prepared statement associated with the <i>hstmt</i> contained a character or binary parameter or literal, but the value exceeded the maximum length of the associated table column.  The prepared statement associated with the <i>hstmt</i> contained a numeric parameter or literal, but the fractional part of the value was truncated.  The prepared statement associated with the <i>hstmt</i> contained a date or time parameter or literal, and a time-stamp value was truncated.
01006	Privilege not revoked	The prepared statement associated with the <i>hstmt</i> was REVOKE, but the user did not have the specified privilege (function returns SQL_SUCCESS_WITH_INFO).
01S03	No rows updated or deleted	The prepared statement associated with the <i>hstmt</i> was a positioned UPDATE or DELETE statement, but no rows were updated or deleted (function returns SQL_SUCCESS_WITH_INFO).
01S04	More than one row updated or deleted	The prepared statement associated with the <i>hstmt</i> was a positioned UPDATE or DELETE statement, and more than one row was updated or deleted (function returns SQL_SUCCESS_WITH_INFO).
07001	Wrong number of parameters	The number of parameters specified in <b>SQLBindParameter</b> was less than the number of parameters in the prepared statement associated with the <i>hstmt</i> .
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.

(1 of 4)

SQLSTATE	Error	Description
22003	Numeric value out of range	The prepared statement associated with the <i>hstmt</i> contained a numeric parameter, and the parameter value caused the whole (as opposed to fractional) part of the number to be truncated when assigned to the associated table column.
22005	Error in assignment	The prepared statement associated with the <i>hstmt</i> contained a parameter, but the value was incompatible with the data type of the associated table column.
22008	Datetime field overflow	The prepared statement associated with the <i>hstmt</i> contained a date, time, or time-stamp parameter or literal, and the value was an invalid date, time, or time stamp, respectively.
22012	Division by zero	The prepared statement associated with the <i>hstmt</i> contained an arithmetic expression that caused division by zero.
23000	Integrity constraint violation	The prepared statement associated with the <i>hstmt</i> contained a parameter. The parameter value was NULL for a column defined as NOT NULL in the associated table column, a duplicate value was supplied for a column constrained to contain only unique values, or some other integrity constraint was violated.
24000	Invalid cursor state	A cursor was already opened on the statement handle.  The prepared statement associated with the <i>hstmt</i> contained a positioned update or delete statement, and the cursor was positioned before the start of the result set or after the end of the result set.
40001	Serialization failure	The transaction to which the prepared statement associated with the <i>hstmt</i> belonged was terminated to prevent deadlock.

(2 of 4)

SQLSTATE	Error	Description
42000	Syntax error or access violation	The user did not have permission to execute the prepared statement associated with the <i>hstmt</i> .
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	<p>(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) The <i>hstmt</i> was not prepared. Either the <i>hstmt</i> was not in an executed state, or a cursor was open on the <i>hstmt</i> and <b>SQLFetch</b> or <b>SQLExtendedFetch</b> had been called.</p> <p>The <i>hstmt</i> was not prepared. It was in an executed state, and either no result set was associated with the <i>hstmt</i> or <b>SQLFetch</b> or <b>SQLExtendedFetch</b> had not been called.</p>

(3 of 4)

SQLSTATE	Error	Description
S1090	Invalid string or buffer length	<p>A parameter value, set with <b>SQLBindParameter</b>, was a null pointer, and the parameter length value was not 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>A parameter value, set with <b>SQLBindParameter</b>, was not a null pointer, and the parameter length value was less than 0, but was not SQL_NTS, SQL_NULL_DATA, or SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p>
S1109	Invalid cursor position	The prepared statement was a positioned UPDATE or DELETE statement, and the cursor was positioned (by <b>SQLExtendedFetch</b> ) on a row for which the value in the <i>rgfRowStatus</i> array in <b>SQLExtendedFetch</b> was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(4 of 4)

**SQLExecute** can return any SQLSTATE that can be returned by **SQLPrepare** based on when the data source evaluates the SQL statement associated with the *hstmt*.



## Usage

**SQLExecute** executes a statement prepared by **SQLPrepare**. Once the application processes or discards the results from a call to **SQLExecute**, the application can call **SQLExecute** again with new parameter values.

To execute a **SELECT** statement more than once, the application must call **SQLFreeStmt** with the **SQL\_CLOSE** parameter before it reissues the **SELECT** statement.

If the data source is in manual-commit mode (requiring explicit transaction initiation), and a transaction has not already been initiated, the driver initiates a transaction before it sends the SQL statement.

If an application uses **SQLPrepare** to prepare and **SQLExecute** to submit a **COMMIT** or **ROLLBACK** statement, it is not interoperable between DBMS products. To commit or roll back a transaction, call **SQLTransact**.

If **SQLExecute** encounters a data-at-execution parameter, it returns **SQL\_NEED\_DATA**. The application sends the data using **SQLParamData** and **SQLPutData**. For more information, see **SQLBindParameter**, **SQLParamData**, **SQLParamOptions**, and **SQLPutData**.

## Code Example

See **SQLBindParameter**, **SQLParamOptions**, and **SQLPutData**.

## Related Functions

<b>For information about</b>	<b>See</b>
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Freeing a statement handle	<b>SQLFreeStmt</b>
Returning a cursor name	<b>SQLGetCursorName</b>
Fetching part or all of a column of data	<b>SQLGetData</b>
Returning the next parameter to send data for	<b>SQLParamData</b>
Preparing a statement for execution	<b>SQLPrepare</b>
Sending parameter data at execution time	<b>SQLPutData</b>
Setting a cursor name	<b>SQLSetCursorName</b>
Setting a statement option	<b>SQLSetStmtOption</b>
Executing a commit or rollback operation	<b>SQLTransact</b>

## SQLExtendedFetch

**SQLExtendedFetch** extends the functionality of **SQLFetch** in the following ways:

- It returns rowset data (one or more rows), in the form of an array, for each bound column.
- It scrolls through the result set according to the setting of a scroll-type argument.

**SQLExtendedFetch** works in conjunction with **SQLSetStmtOption**.

To fetch one row of data at a time in a forward direction, an application should call **SQLFetch**.

For more information about scrolling through result sets, see [“Using Block and Scrollable Cursors” on page 7-9](#).

### Syntax

```
RETCODE SQLExtendedFetch(hstmt, fFetchType, irow, pcrow,
    rgfRowStatus)
```

The **SQLExtendedFetch** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
UWORD	<i>fFetchType</i>	Input	Type of fetch
SDWORD	<i>irow</i>	Input	Number of the row to fetch
UDWORD FAR *	<i>pcrow</i>	Output	Number of rows actually fetched
UWORD FAR *	<i>rgfRowStatus</i>	Output	An array of status values

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLExtendedFetch** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLExtendedFetch** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The data returned for one or more columns was truncated. String values are right truncated. For numeric values, the fractional part of number was truncated (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
01S01	Error in row	An error occurred while fetching one or more rows (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
07006	Restricted data type attribute violation	A data value could not be converted to the C data type specified by <i>fCType</i> in <code>SQLBindCol</code> .
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
22002	Indicator value required but not supplied	If the column value for any bound column is null, the <code>INDICATOR_PTR</code> field of the corresponding descriptor record must not be a null pointer.

(1 of 4)

SQLSTATE	Error	Description
22003	Numeric value out of range	<p>Returning the numeric value (as numeric or string) for one or more columns would have caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>Returning the binary value for one or more columns would have caused a loss of binary significance.</p> <p>For more information, see <a href="#">Appendix C</a>.</p>
22005	Error in assignment	A zero-length string was inserted into a string field, and the string was bound to a numeric data type, so the string was converted to a zero.
22008	Datetime field overflow	An SQL_C_TIME, SQL_C_DATE, or SQL_C_TIMESTAMP value was converted to an SQL_CHAR data type, and the value was an invalid date, time, or time stamp, respectively.
22012	Division by zero	A value from an arithmetic expression was returned which resulted in division by zero.
24000	Invalid cursor state	The <i>hstmt</i> was in an executed state, but no result set was associated with the <i>hstmt</i> .
40001	Serialization failure	The transaction (in which the fetch was executed) was terminated to prevent deadlock.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

(2 of 4)

SQLSTATE	Error	Description
S1002	Invalid column number	<p>A column number specified in the binding for one or more columns was greater than the number of columns in the result set.</p> <p>Column 0 was bound with <b>SQLBindCol</b> and the <code>SQL_USE_BOOKMARKS</code> statement option was set to <code>SQL_UB_OFF</code>.</p>
S1008	Operation canceled	<p>The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.</p>
S1010	Function-sequence error	<p>(DM) The specified <i>hstmt</i> was not in an executed state. The function was called without first calling <b>SQLExecDirect</b>, <b>SQLExecute</b>, or a catalog function.</p> <p>(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b>, was called for the <i>hstmt</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>(DM) <b>SQLExtendedFetch</b> was called for an <i>hstmt</i> after <b>SQLFetch</b> was called and before <b>SQLFreeStmt</b> was called with the <code>SQL_CLOSE</code> option.</p>
S1106	Fetch type out of range	<p>(DM) The value specified for the argument <i>fFetchType</i> was invalid.</p> <p>The value of the <code>SQL_CURSOR_TYPE</code> statement option was <code>SQL_CURSOR_FORWARD_ONLY</code> and the value of argument <i>fFetchType</i> was not <code>SQL_FETCH_NEXT</code>.</p>

(3 of 4)

SQLSTATE	Error	Description
S1107	Row value out of range	The value specified with the statement option <code>SQL_CURSOR_TYPE</code> was <code>SQL_CURSOR_KEYSET_DRIVEN</code> , but the value specified with the <code>SQL_KEYSET_SIZE</code> statement option was greater than 0 and less than the value specified with the <code>SQL_ROWSET_SIZE</code> statement option.
S1C00	Driver not capable	<p>The driver or data source does not support the specified fetch type.</p> <p>The driver or data source does not support the conversion specified by the combination of the <i>fCType</i> in <code>SQLBindCol</code> and the SQL data type of the corresponding column. This error applies only when the SQL data type of the column was mapped to a driver-specific SQL data type.</p> <p>The argument <i>fFetchType</i> was <code>SQL_FETCH_RESUME</code>, and the driver supports ODBC 2.0.</p>
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <code>SQLSetStmtOption</code> , <code>SQL_QUERY_TIMEOUT</code> .

(4 of 4)

## Usage

**SQLExtendedFetch** returns one rowset of data to the application. An application cannot mix calls to **SQLExtendedFetch** and **SQLFetch** for the same cursor.

An application specifies the number of rows in the rowset by calling **SQLSetStmtOption** with the `SQL_ROWSET_SIZE` statement option.

## **Binding**

If any columns in the result set have been bound with **SQLBindCol**, the driver converts the data for the bound columns as necessary and stores it in the locations bound to those columns. The result set can be bound in a column-wise (the default) or row-wise fashion.

### *Column-Wise Binding*

To bind a result set in column-wise fashion, an application specifies **SQL\_BIND\_BY\_COLUMN** for the **SQL\_BIND\_TYPE** statement option; this is the default value.

#### **To bind each column**

1. The application allocates an array of data-storage buffers. The array has as many elements as the rowset has rows, plus an additional element if the application searches for key values or appends new rows of data. The size of each buffer is the maximum size of the C data that can be returned for the column. For example, when the C data type is **SQL\_C\_DEFAULT**, the size of each buffer is the column length. When the C data type is **SQL\_C\_CHAR**, the size of each buffer is the display size of the data. For more information, see [“Converting Data from SQL to C Data Types” on page C-13](#) and [“Precision, Scale, Length, and Display Size” on page C-8](#).
2. The application allocates an array of **SDWORDS** to hold the number of bytes available to return for each row in the column. The array has as many elements as the rowset has rows.
3. The application calls **SQLBindCol**:
  - The *rgbValue* argument specifies the address of the data-storage array.
  - The *cbValueMax* argument specifies the size of each buffer in the data-storage array.
  - The *pcbValue* argument specifies the address of the number-of-bytes array.



When the application calls **SQLExtendedFetch**, the driver retrieves the data and the number of bytes that are available to return and stores them in the buffers that are allocated by the application:

- For each bound column, the driver stores the data in the *rgbValue* buffer bound to the column. It stores the first row of data at the start of the buffer and each subsequent row of data at an offset of *cbValueMax* bytes from the data for the previous row.
- For each bound column, the driver stores the number of bytes that are available to return in the *pcbValue* buffer that is bound to the column. This is the number of bytes available before calling **SQLExtendedFetch**. (If the number of bytes available to return cannot be determined in advance, the driver sets *pcbValue* to `SQL_NO_TOTAL`. If the data for the column is `NULL`, the driver sets *pcbValue* to `SQL_NULL_DATA`.) It stores the number of bytes available to return for the first row at the start of the buffer and the number of bytes available to return for each subsequent row at an offset of `sizeof(SDWORD)` from the value for the previous row.

### *Row-Wise Binding*

To bind a result set in row-wise fashion, an application must specify the `SQL_BIND_TYPE` statement option for **SQLSetStmtOption**.

#### **To bind a result set in row-wise fashion**

1. The application declares a structure that can hold a single row of retrieved data and the associated data lengths. For each bound column, the structure contains one field for the data and one `SDWORD` field for the number of bytes that are available to return. The size of the data field is the maximum size of the C data that can be returned for the column.
2. The application calls **SQLSetStmtOption** with *fOption* set to `SQL_BIND_TYPE` and *vParam* set to the size of the structure.
3. The application allocates an array of these structures. The array has as many elements as the rowset has rows, plus an additional element if the application searches for key values or appends new rows of data.

4. The application calls **SQLBindCol** for each column to be bound:
  - The *rgbValue* argument specifies the address of the data field of the column in the first array element.
  - The *cbValueMax* argument specifies the size of the data field of the column.
  - The *pcbValue* argument specifies the address of the number-of-bytes field of the column in the first array element.

When the application calls **SQLExtendedFetch**, the driver retrieves the data and the number of bytes that are available to return and stores them in the buffers that are allocated by the application:

- For each bound column, the driver stores the first row of data at the address specified by *rgbValue* for the column and each subsequent row of data at an offset of *vParam* bytes from the data for the previous row.
- For each bound column, the driver stores the number of bytes that are available to return for the first row at the address specified by *pcbValue* and the number of bytes that are available to return for each subsequent row at an offset of *vParam* bytes from the value for the previous row. This is the number of bytes that are available prior to calling **SQLExtendedFetch**. (If the number of bytes that are available to return cannot be determined in advance, the driver sets *pcbValue* to `SQL_NO_TOTAL`. If the data for the column is NULL, the driver sets *pcbValue* to `SQL_NULL_DATA`.)

### ***Positioning the Cursor***

The following operations require a cursor position:

- Positioned update and delete statements
- Calls to **SQLGetData**

Before the application executes a positioned update, a delete statement, or a call to **SQLGetData**, it must position the cursor by calling **SQLExtendedFetch** to retrieve a rowset; the cursor points to the first row in the rowset.

The following table shows the rowset and code returned when the application requests different rowsets.

Requested Rowset	Return Code	Cursor Position	Returned Rowset
Before start of result set	SQL_NO_DATA_FOUND	Before start of result set	None. The contents of the rowset buffers are undefined.
Overlaps start of result set	SQL_SUCCESS	Row 1 of rowset	First rowset in result set.
Within result set	SQL_SUCCESS	Row 1 of rowset	Requested rowset.
Overlaps end of result set	SQL_SUCCESS	Row 1 of rowset	For rows in the rowset that overlap the result set, data is returned.  For rows in the rowset outside the result set, the contents of the <i>rgbValue</i> and <i>pcbValue</i> buffers are undefined, and the <i>rgfRowStatus</i> array contains SQL_ROW_NOROW.
After end of result set	SQL_NO_DATA_FOUND	After end of result set	None. The contents of the rowset buffers are undefined.

For example, suppose a result set has 100 rows, and the rowset size is 5. The following table shows the rowset and code returned by **SQLExtendedFetch** for different values of *irow* when the fetch type is SQL\_FETCH\_RELATIVE.

Current Rowset	irow	Return Code	New Rowset
1 to 5	-5	SQL_NO_DATA_FOUND	None.
1 to 5	-3	SQL_SUCCESS	1 to 5.
96 to 100	5	SQL_NO_DATA_FOUND	None.
96 to 100	3	SQL_SUCCESS	99 and 100. For rows 3, 4, and 5 in the rowset, the <i>rgfRowStatusArray</i> is set to SQL_ROW_NOROW.

Before **SQLExtendedFetch** is called the first time, the cursor is positioned before the start of the result set.

For the purpose of moving the cursor, deleted rows (that is, rows with an entry in the *rgfRowStatus* array of SQL\_ROW\_DELETED) are treated no differently than other rows. For example, calling **SQLExtendedFetch** with *fFetchType* set to SQL\_FETCH\_ABSOLUTE and *irow* set to 15 returns the rowset starting at row 15, even if the *rgfRowStatus* array for row 15 is SQL\_ROW\_DELETED.

### ***Processing Errors***

If an error occurs that pertains to the entire rowset, such as SQLSTATE S1T00 (Time-out expired), the driver returns SQL\_ERROR and the appropriate SQLSTATE. The contents of the rowset buffers are undefined, and the cursor position is unchanged.

If an error occurs that pertains to a single row, the driver performs the following actions:

- Sets the element in the *rgfRowStatus* array for the row to SQL\_ROW\_ERROR
- Posts SQLSTATE 01S01 (Error in row) in the error queue
- Posts zero or more additional SQLSTATE values for the error after SQLSTATE 01S01 (Error in row) in the error queue

After it processes the error or warning, the driver continues the operation for the remaining rows in the rowset and returns SQL\_SUCCESS\_WITH\_INFO. Thus, for each error that pertains to a single row, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATES.

After the driver processes the error, it fetches the remaining rows in the rowset and returns SQL\_SUCCESS\_WITH\_INFO. Thus, for each row that returns an error, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATE values.

If the rowset contains rows that have already been fetched, the driver is not required to return SQLSTATE values for errors that occurred when the rows were first fetched. However, it is required to return SQLSTATE 01S01 (Error in row) for each row in which an error originally occurred and to return SQL\_SUCCESS\_WITH\_INFO. For example, a static cursor that maintains a cache might cache row-status information (so it can determine which rows contain errors) but might not cache the SQLSTATE associated with those errors.

Error rows do not affect relative cursor movements. For example, suppose the result set size is 100, and the rowset size is 10. If the current rowset is rows 11 through 20 and the element in the *rgfRowStatus* array for row 11 is SQL\_ROW\_ERROR, calling **SQLExtendedFetch** with the SQL\_FETCH\_NEXT fetch type still returns rows 21 through 30.

If the driver returns any warnings, such as SQLSTATE 01004 (Data truncated), it returns warnings that apply to the entire rowset or to unknown rows in the rowset before it returns error information that applies to specific rows. It returns warnings for specific rows along with any other error information about those rows.

### ***fFetchType* Argument**

The *fFetchType* argument specifies how to move through the result set. It is one of the following values:

- SQL\_FETCH\_NEXT
- SQL\_FETCH\_FIRST
- SQL\_FETCH\_LAST
- SQL\_FETCH\_PRIOR
- SQL\_FETCH\_ABSOLUTE
- SQL\_FETCH\_RELATIVE

If the value of the SQL\_CURSOR\_TYPE statement option is SQL\_CURSOR\_FORWARD\_ONLY, the *fFetchType* argument must be SQL\_FETCH\_NEXT.

***Important:*** If an application specifies SQL\_FETCH\_RESUME, the driver manager returns SQLSTATE S1C00 (Driver not capable). The SQL\_FETCH\_RESUME fetch type is obsolete in ODBC 2.x drivers.



*Moving by Row Position*

**SQLExtendedFetch** supports the following values of the *fFetchType* argument to move the cursor relative to the current rowset.

<i>fFetchType</i> Argument	Action
SQL_FETCH_NEXT	The driver returns the next rowset. If the cursor is positioned before the start of the result set, this is equivalent to SQL_FETCH_FIRST.
SQL_FETCH_PRIOR	The driver returns the prior rowset. If the cursor is positioned after the end of the result set, this is equivalent to SQL_FETCH_LAST.
SQL_FETCH_RELATIVE	The driver returns the rowset <i>irow</i> rows from the start of the current rowset. If <i>irow</i> equals 0, the driver refreshes the current rowset. If the cursor is positioned before the start of the result set and <i>irow</i> is greater than 0, or if the cursor is positioned after the end of the result set and <i>irow</i> is less than 0, this is equivalent to SQL_FETCH_ABSOLUTE.

It supports the following values of the *fFetchType* argument to move the cursor to an absolute position in the result set.

<i>fFetchType</i> Argument	Action
SQL_FETCH_FIRST	The driver returns the first rowset in the result set.
SQL_FETCH_LAST	The driver returns the last complete rowset in the result set.
SQL_FETCH_ABSOLUTE	<p>If <i>irow</i> is greater than 0, the driver returns the rowset starting at row <i>irow</i>.</p> <p>If <i>irow</i> equals 0, the driver returns SQL_NO_DATA_FOUND, and the cursor is positioned before the start of the result set.</p> <p>If <i>irow</i> is less than 0, the driver returns the rowset starting at row <math>n+irow+1</math>, where <i>n</i> is the number of rows in the result set. For example, if <i>irow</i> is -1, the driver returns the rowset that starts at the last row in the result set. If the result set size is 10 and <i>irow</i> is -10, the driver returns the rowset that starts at the first row in the result set.</p>

### ***row Argument***

For the SQL\_FETCH\_ABSOLUTE fetch type, **SQLExtendedFetch** returns the rowset that starts at the row number specified by the *row* argument.

For the SQL\_FETCH\_RELATIVE fetch type, **SQLExtendedFetch** returns the rowset that starts *row* rows from the first row in the current rowset.

The *row* argument is ignored for the SQL\_FETCH\_NEXT, SQL\_FETCH\_PRIOR, SQL\_FETCH\_FIRST, and SQL\_FETCH\_LAST fetch types.

### ***rgfRowStatus Argument***

In the *rgfRowStatus* array, **SQLExtendedFetch** returns any changes in status to each row since it was last retrieved from the data source. Rows might be unchanged (SQL\_ROW\_SUCCESS), updated (SQL\_ROW\_UPDATED), deleted (SQL\_ROW\_DELETED), added (SQL\_ROW\_ADDED), or unretrievable due to an error (SQL\_ROW\_ERROR). For static cursors, this information is available for all rows. For keyset, mixed, and dynamic cursors, this information is available only for rows in the keyset; the driver does not save data outside the keyset and cannot compare the newly retrieved data to anything.

***Important:*** *The Informix driver cannot detect changes to data.*

*The number of elements must equal the number of rows in the rowset (as defined by the SQL\_ROWSET\_SIZE statement option). If the number of rows fetched is less than the number of elements in the status array, the driver sets remaining status elements to SQL\_ROW\_NOROW.*

For keyset, mixed, and dynamic cursors, if a key value is updated, the row of data is considered to have been deleted, and a new row is added.



## Code Example

The following two examples show how an application could use column-wise or row-wise binding to bind storage locations to the same result set.

### *Column-Wise Binding*

In the following example, an application declares storage locations for column-wise bound data and the returned numbers of bytes. Because column-wise binding is the default, there is no need to request column-wise binding with **SQLSetStmtOption**, as in the row-wise binding example. However, the application does call **SQLSetStmtOption** to specify the number of rows in the rowset.

The application then executes a **SELECT** statement to return a result set of the employee names and birthdays, which is sorted by birthday. It calls **SQLBindCol** to bind the columns of data, passing the addresses of storage locations for both the data and the returned numbers of bytes. Finally, the application fetches the rowset data with **SQLExtendedFetch** and prints each employee's name and birthday.

```
#define ROWS 100
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR      szName[ROWS][NAME_LEN], szBirthday[ROWS][BDAY_LEN];
SWORD      sAge[ROWS];
SDWORD     cbName[ROWS], cbAge[ROWS], cbBirthday[ROWS];

UDWORD     crow, irow;
UWORD      rgfRowStatus[ROWS];

SQLSetStmtOption(hstmt, SQL_CONCURRENCY, SQL_CONCUR_READ_ONLY);
SQLSetStmtOption(hstmt, SQL_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN);
SQLSetStmtOption(hstmt, SQL_ROWSET_SIZE, ROWS);
retcode = SQLExecDirect(hstmt,

        "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE ORDER BY 3, 2, 1",
        SQL_NTS);

if (retcode == SQL_SUCCESS) {
    SQLBindCol(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, cbName);
    SQLBindCol(hstmt, 2, SQL_C_SSHORT, sAge, 0, cbAge);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, szBirthday, BDAY_LEN,
        cbBirthday);

    /* Fetch the rowset data and print each row. */
```



```

/* On an error, display a message and exit. */
while (TRUE) {
    retcode = SQLExtendedFetch(hstmt, SQL_FETCH_NEXT, 1, &crow,
                               rgfRowStatus);
    if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
        show_error();
    }
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
        for (irow = 0; irow < crow; irow++) {
            if (rgfRowStatus[irow] != SQL_ROW_DELETED &&

                rgfRowStatus[irow] != SQL_ROW_ERROR)
                fprintf(out, "%-*s  %-2d  %*s",

                        NAME_LEN-1, szName[irow], sAge[irow],

                        BDAY_LEN-1, szBirthday[irow]);

            }
        } else {
            break;
        }
    }
}

```

### ***Row-Wise Binding***

In the following example, an application declares an array of structures to hold row-wise bound data and the returned numbers of bytes. Using **SQLSetStmtOption**, it requests row-wise binding and passes the size of the structure to the driver. The driver uses this size to find successive storage locations in the array of structures. Using **SQLSetStmtOption**, it specifies the size of the rowset.

The application then executes a **SELECT** statement to return a result set of the employee names and birthdays, which is sorted by birthday. It calls **SQLBindCol** to bind the columns of data, passing the addresses of storage locations for both the data and the returned numbers of bytes. Finally, the application fetches the rowset data with **SQLExtendedFetch** and prints each employee's name and birthday.

```
#define ROWS 100
#define NAME_LEN 30
#define BDAY_LEN 11

typedef struct {
    UCHAR    szName[NAME_LEN];
    SDWORD  cbName;
    SWORD   sAge;
    SDWORD  cbAge;
    UCHAR    szBirthday[BDAY_LEN];
    SDWORD  cbBirthday;
} EmpTable;

EmpTable rget[ROWS];
UDWORD   crow, irow;
UWORD    rgfRowStatus[ROWS];

SQLSetStmtOption(hstmt, SQL_BIND_TYPE, sizeof(EmpTable));
SQLSetStmtOption(hstmt, SQL_CONCURRENCY, SQL_CONCUR_READ_ONLY);
SQLSetStmtOption(hstmt, SQL_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN);
SQLSetStmtOption(hstmt, SQL_ROWSET_SIZE, ROWS);
retcode = SQLExecDirect(hstmt,
    "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE ORDER BY 3, 2, 1",
    SQL_NTS);

if (retcode == SQL_SUCCESS) {
    SQLBindCol(hstmt, 1, SQL_C_CHAR, rget[0].szName, NAME_LEN,
        &rget[0].cbName);
    SQLBindCol(hstmt, 2, SQL_C_SSHORT, &rget[0].sAge, 0,
        &rget[0].cbAge);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, rget[0].szBirthday, BDAY_LEN,
        &rget[0].cbBirthday);

    /* Fetch the rowset data and print each row. */
    /* On an error, display a message and exit. */

    while (TRUE) {
        retcode = SQLExtendedFetch(hstmt, SQL_FETCH_NEXT, 1, &crow,
            rgfRowStatus);
        if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
            show_error();
        }
        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
            for (irow = 0; irow < crow; irow++) {
                if (rgfRowStatus[irow] != SQL_ROW_DELETED &&
                    rgfRowStatus[irow] != SQL_ROW_ERROR)

```

```

        fprintf(out, "%-*s %-2d %*s",
                NAME_LEN-1, rget[irow].szName, rget[irow].sAge,
                BDAY_LEN-1, rget[irow].szBirthday);
    }
} else {
    break;
}
}
}

```

## Related Function

The following table shows where to find information about related functions.

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning information about a column in a result set	<b>SQLDescribeCol</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Returning the number of result set columns	<b>SQLNumResultCols</b>
Setting a statement option	<b>SQLSetStmtOption</b>

---

## SQLFetch

**SQLFetch** fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with **SQLBindCol**.

### Syntax

```
RETCODE SQLFetch(hstmt)
```

The **SQLFetch** function accepts the following argument.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLFetch** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLFetch** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The data returned for one or more columns was truncated. String values are right truncated. For numeric values, the fractional part of number was truncated (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
07006	Restricted data-type attribute violation	The data value could not be converted to the data type specified by <i>fCType</i> in <b>SQLBindCol</b> .
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
22002	Indicator value required but not supplied	If the column value for any bound column is null, the <code>INDICATOR_PTR</code> field of the corresponding descriptor record must not be a null pointer.

(1 of 3)

SQLSTATE	Error	Description
22003	Numeric value out of range	<p>Returning the numeric value (as numeric or string) for one or more columns would have caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>Returning the binary value for one or more columns would have caused a loss of binary significance.</p> <p>For more information, see <a href="#">“Converting Data from SQL to C Data Types”</a> on page C-13.</p>
22005	Error in assignment	A zero-length string was inserted into a string field, and the string was bound to a numeric data type, so the string was converted to a zero.
22008	Datetime field overflow	An SQL_C_TIME, SQL_C_DATE, or SQL_C_TIMESTAMP value was converted to an SQL_CHAR data type, and the value was an invalid date, time, or time stamp, respectively.
22012	Division by zero	A value from an arithmetic expression was returned that resulted in division by zero.
24000	Invalid cursor state	The <i>hstmt</i> was in an executed state, but no result set was associated with the <i>hstmt</i> .
40001	Serialization failure	The transaction in which the fetch was executed was terminated to prevent deadlock.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.

(2 of 3)

SQLSTATE	Error	Description
S1002	Invalid column number	A column number specified in the binding for one or more columns was greater than the number of columns in the result set.  A column number specified in the binding for a column was 0..
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) The specified <i>hstmt</i> was not in an executed state. The function was called without first calling <b>SQLExecDirect</b> , <b>SQLExecute</b> , or a catalog function.  (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.  (DM) <b>SQLExtendedFetch</b> was called for an <i>hstmt</i> after <b>SQLFetch</b> was called and before <b>SQLFreeStmt</b> was called with the SQL_CLOSE option.
S1C00	Driver not capable	The driver or data source does not support the conversion specified by the combination of the <i>fCType</i> in <b>SQLBindCol</b> and the SQL data type of the corresponding column. This error applies only when the SQL data type of the column was mapped to a driver-specific SQL data type.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(3 of 3)

## Usage

**SQLFetch** positions the cursor on the next row of the result set. Before **SQLFetch** is called the first time, the cursor is positioned before the start of the result set. When the cursor is positioned on the last row of the result set, **SQLFetch** returns `SQL_NO_DATA_FOUND`, and the cursor is positioned after the end of the result set. An application cannot mix calls to **SQLExtendedFetch** and **SQLFetch** for the same cursor.

If the application called **SQLBindCol** to bind columns, **SQLFetch** stores data into the locations specified by the calls to **SQLBindCol**. If the application does not call **SQLBindCol** to bind any columns, **SQLFetch** does not return any data; it moves the cursor to the next row. An application can call **SQLGetData** to retrieve data that is not bound to a storage location.

The driver manages cursors during the fetch operation and places each value of a bound column into the associated storage. The driver follows these guidelines when it performs a fetch operation:

- **SQLFetch** accesses column data in left-to-right order.
- After each fetch, *pcbValue* (specified in **SQLBindCol**) contains the number of bytes that are available to return for the column. This is the number of bytes that are available prior to calling **SQLFetch**. If the number of bytes that are available to return cannot be determined in advance, the driver sets *pcbValue* to `SQL_NO_TOTAL`. (If `SQL_MAX_LENGTH` has been specified with **SQLSetStmtOption** and the number of bytes that are available to return is greater than `SQL_MAX_LENGTH`, *pcbValue* contains `SQL_MAX_LENGTH`.)



*Tip: The `SQL_MAX_LENGTH` statement option is intended to reduce network traffic and might not be supported by all drivers. To guarantee that data is truncated, an application should allocate a buffer of the desired size and specify this size in the *cbValueMax* argument.*

- If *rgbValue* cannot hold the entire result, the driver stores part of the value and returns `SQL_SUCCESS_WITH_INFO`. A subsequent call to **SQLError** indicates that a truncation occurred. The application can compare *pcbValue* to *cbValueMax* (specified in **SQLBindCol**) to determine which column or columns were truncated. If *pcbValue* is greater than or equal to *cbValueMax*, then truncation occurred.
- If the data value for the column is `NULL`, the driver stores `SQL_NULL_DATA` in *pcbValue*.



**SQLFetch** is valid only after a call that returns a result set.

For information about conversions allowed by **SQLBindCol** and **SQLGetData**, see [“Converting Data from SQL to C Data Types”](#) on page C-13.

## Code Example

See **SQLBindCol**, **SQLColumns**, **SQLGetData**, and **SQLProcedures**.

## Related Functions.

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning information about a column in a result set	<b>SQLDescribeCol</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Freeing a statement handle	<b>SQLFreeStmt</b>
Fetching part or all of a column of data	<b>SQLGetData</b>
Returning the number of result set columns	<b>SQLNumResultCols</b>
Preparing a statement for execution	<b>SQLPrepare</b>

---

## SQLFreeConnect

**SQLFreeConnect** releases a connection handle and frees all memory associated with the handle.

### Syntax

```
RETCODE SQLFreeConnect(hdbc)
```

The **SQLFreeConnect** function accepts the following argument.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLFreeConnect** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLFreeConnect** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1010	Function-sequence error	(DM) The function was called prior to calling <b>SQLDisconnect</b> for the <i>hdbc</i> .

## Usage

Prior to calling **SQLFreeConnect**, an application must call **SQLDisconnect** for the *hdbc*. Otherwise, **SQLFreeConnect** returns `SQL_ERROR`, and the *hdbc* remains valid. **SQLDisconnect** automatically drops any *hstmts* open on the *hdbc*.

## Code Example

See **SQLBrowseConnect** and **SQLConnect**.

## Related Functions

For information about	See
Allocating a statement handle	<b>SQLAllocConnect</b>
Connecting to a data source	<b>SQLConnect</b>
Disconnecting from a data source	<b>SQLDisconnect</b>
Connecting to a data source using a connection string or dialog box	<b>SQLDriverConnect</b>
Freeing an environment handle	<b>SQLFreeEnv</b>
Freeing a statement handle	<b>SQLFreeStmt</b>

## SQLFreeEnv

**SQLFreeEnv** frees the environment handle and releases all memory associated with the environment handle.

### Syntax

```
RETCODE SQLFreeEnv(henv)
```

The **SQLFreeEnv** function accepts the following argument.

Type	Argument	Use	Description
HENV	<i>henv</i>	Input	Environment handle

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLFreeEnv** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLFreeEnv** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1010	Function-sequence error	(DM) There was at least one <i>hdbc</i> in an allocated or connected state. Call <b>SQLDisconnect</b> and <b>SQLFreeConnect</b> for each <i>hdbc</i> before calling <b>SQLFreeEnv</b> .

## Usage

Before an application calls **SQLFreeEnv**, it must call **SQLFreeConnect** for any *hdbc* allocated under the *henv*. Otherwise, **SQLFreeEnv** returns `SQL_ERROR` and the *henv* and any active *hdbc* remains valid.

When the driver manager processes the **SQLFreeEnv** function, it checks the `TraceAutoStop` keyword in the ODBC section of the `odbc.ini` file. If the keyword is set to 1, the driver manager disables tracing for all applications and sets the `Trace` keyword in the ODBC section of the `odbc.ini` file to 0.

## Code Example

See **SQLBrowseConnect** and **SQLConnect**.

## Related Functions

---

For information about	See
Allocating an environment handle	<b>SQLAllocEnv</b>
Freeing a connection handle	<b>SQLFreeConnect</b>

---

## SQLFreeStmt

**Core**

**SQLFreeStmt** stops processing associated with a specific *hstmt*, closes any open cursors associated with the *hstmt*, discards pending results, and, optionally, frees all resources associated with the statement handle.

### Syntax

```
RETCODE SQLFreeStmt(hstmt, fOption)
```

The **SQLFreeStmt** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>fOption</i>	Input	<p>One of the following options:</p> <p>SQL_CLOSE: Close the cursor associated with <i>hstmt</i> (if one is defined) and discard all pending results. The application can reopen this cursor later by executing a SELECT statement again with the same or different parameter values. If no cursor is open, this option has no effect for the application.</p> <p>SQL_DROP: Release the <i>hstmt</i>, free all resources associated with it, close the cursor (if one is open), and discard all pending rows. This option terminates all access to the <i>hstmt</i>. The <i>hstmt</i> must be reallocated to be reused.</p> <p>SQL_UNBIND: Release all column buffers bound by <b>SQLBindCol</b> for the given <i>hstmt</i>.</p> <p>SQL_RESET_PARAMS: Release all parameter buffers set by <b>SQLBindParameter</b> for the given <i>hstmt</i>.</p>

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLFreeStmt** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLFreeStmt** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1092	Option type out of range	(DM) The value specified for the argument <i>fOption</i> was not:  <code>SQL_CLOSE</code> <code>SQL_DROP</code> <code>SQL_UNBIND</code> <code>SQL_RESET_PARAMS</code>



## Usage

An application can call **SQLFreeStmt** to terminate processing of a SELECT statement with or without canceling the statement handle.

The SQL\_DROP option frees all resources that are allocated by the **SQLAllocStmt** function.

## Code Example

See **SQLBrowseConnect** and **SQLConnect**.

## Related Functions

---

For information about	See
Allocating a statement handle	<b>SQLAllocStmt</b>
Canceling statement processing	<b>SQLCancel</b>
Setting a cursor name	<b>SQLSetCursorName</b>

---

## SQLGetConnectOption

**SQLGetConnectOption** returns the current setting of a connection option.

### Syntax

```
RETCODE SQLGetConnectOption(hdbc, fOption, pvParam)
```

The **SQLGetConnectOption** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
UWORD	<i>fOption</i>	Input	Option to retrieve.
PTR	<i>pvParam</i>	Output	Value associated with <i>fOption</i> . Depending on the value of <i>fOption</i> , a 32-bit integer value or a pointer to a null-terminated character string will be returned in <i>pvParam</i> .

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetConnectOption** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetConnectOption** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection not open	(DM) An <i>fOption</i> value was specified that required an open connection.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) <b>SQLBrowseConnect</b> was called for the <i>hdbc</i> and returned <code>SQL_NEED_DATA</code> . This function was called before <b>SQLBrowseConnect</b> returned <code>SQL_SUCCESS_WITH_INFO</code> or <code>SQL_SUCCESS</code> .
S1092	Option type out of range	(DM) The value specified for the argument <i>fOption</i> was in the block of numbers reserved for ODBC connection and statement options but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The driver or data source does not support the value specified for the argument <i>fOption</i> .



## Usage

For a list of options, see **SQLSetConnectOption**.

**Important:** When *fOption* specifies an option that returns a string, *pvParam* must be a pointer to storage for the string. The maximum length of the string is `SQL_MAX_OPTION_STRING_LENGTH` bytes (excluding the null-termination byte).

Depending on the option, an application does not need to establish a connection prior to calling **SQLGetConnectOption**. However, if **SQLGetConnectOption** is called and the specified option does not have a default and has not been set by a prior call to **SQLSetConnectOption**, **SQLGetConnectOption** returns `SQL_NO_DATA_FOUND`.

Although an application can set statement options using **SQLSetConnectOption**, an application cannot use **SQLGetConnectOption** to retrieve statement-option values; it must call **SQLGetStmtOption** to retrieve the settings of statement options.

## Related Functions

For information about	See
Returning the setting of a statement option	<b>SQLGetStmtOption</b>
Setting a connection option	<b>SQLSetConnectOption</b>
Setting a statement option	<b>SQLSetStmtOption</b>

## Core

## SQLGetCursorName

**SQLGetCursorName** returns the cursor name associated with a specified *hstmt*.

### Syntax

```
RETCODE SQLGetCursorName(hstmt, szCursor, cbCursorMax,
pcbCursor)
```

The **SQLGetCursorName** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szCursor</i>	Output	Pointer to storage for the cursor name.
SWORD	<i>cbCursorMax</i>	Input	Length of <i>szCursor</i> .
SWORD FAR *	<i>pcbCursor</i>	Output	Total number of bytes (excluding the null termination byte) available to return in <i>szCursor</i> . If the number of bytes available to return is greater than or equal to <i>cbCursorMax</i> , the cursor name in <i>szCursor</i> is truncated to <i>cbCursorMax</i> - 1 bytes.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetCursorName** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetCursorName** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The buffer <code>szCursor</code> was not large enough to return the entire cursor name, so the cursor name was truncated. The argument <code>pcbCursor</code> contains the length of the untruncated cursor name (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <code>szErrorMsg</code> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support executing or completing the function.

(1 of 2)

SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1015	No cursor name available	(DM) There was no open cursor on the <i>hstmt</i> , and no cursor name had been set with <b>SQLSetCursorName</b> .
S1090	Invalid string or buffer length	(DM) The value specified in the argument <i>cbCursorMax</i> was less than 0.

(2 of 2)

## Usage

The only INFORMIX-CLI SQL statements that use a cursor name are positioned update and delete (for example, UPDATE *table-name* ...WHERE CURRENT OF *cursor-name*). If the application does not call **SQLSetCursorName** to define a cursor name when a SELECT statement executes, the driver generates a name that begins with the letters SQL\_CUR and does not exceed 18 characters.

**SQLGetCursorName** returns the name of a cursor regardless of whether the name was created explicitly or implicitly.

A cursor name that is set either explicitly or implicitly remains set until the *hstmt* with which it is associated is dropped, using **SQLFreeStmt** with the SQL\_DROP option.

## Related Functions

For information about	See
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Preparing a statement for execution	<b>SQLPrepare</b>
Setting a cursor name	<b>SQLSetCursorName</b>
Setting cursor scrolling options	<b>SQLSetScrollOptions</b>

## SQLGetData

**SQLGetData** returns result data for a single unbound column in the current row. The application must call **SQLFetch** or **SQLExtendedFetch** to position the cursor on a row of data before it calls **SQLGetData**. It is possible to use **SQLBindCol** for some columns and use **SQLGetData** for others within the same row. This function can be used to retrieve character or binary data values in parts from a column with a character, binary, or data-source-specific data type (for example, data from SQL\_LONGVARBINARY or SQL\_LONGVARCHAR columns).

## Syntax

```
RETCODE SQLGetData(hstmt, icol, fCType, rgbValue, cbValueMax,
pcbValue)
```

The **SQLGetData** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>icol</i>	Input	Column number of result data, ordered sequentially left to right, starting at 1.

(1 of 3)



Type	Argument	Use	Description
SWORD	<i>fCType</i>	Input	<p>The C data type of the result data. This must be one of the following values:</p> <p>SQL_C_BINARY  SQL_C_BIT  SQL_C_CHAR  SQL_C_DATE  SQL_C_DEFAULT  SQL_C_DOUBLE  SQL_C_FLOAT  SQL_C_SLONG  SQL_C_SSHORT  SQL_C_STINYINT  SQL_C_TIME  SQL_C_TIMESTAMP  SQL_C_ULONG  SQL_C_USHORT  SQL_C_UTINYINT</p> <p>SQL_C_DEFAULT specifies that data be converted to its default C data type.</p> <p>For information about how data is converted, see <a href="#">“Converting Data from SQL to C Data Types”</a> on page C-13.</p>
PTR	<i>rgbValue</i>	Output	Pointer to storage for the data.
SDWORD	<i>cbValueMax</i>	Input	<p>Maximum length of the <i>rgbValue</i> buffer. For character data, <i>rgbValue</i> must also include space for the null-termination byte.</p> <p>For character and binary C data, <i>cbValueMax</i> determines the amount of data that can be received in a single call to <b>SQLGetData</b>. For all other types of C data, <i>cbValueMax</i> is ignored; the driver assumes that the size of <i>rgbValue</i> is the size of the C data type specified with <i>fCType</i> and returns the entire data value. For more information about length, see <a href="#">“Precision, Scale, Length, and Display Size”</a> on page C-8.</p>

(2 of 3)

Type	Argument	Use	Description
SDWORD FAR *	<i>pcbValue</i>	Output	<p>SQL_NULL_DATA, the total number of bytes (excluding the null-termination byte for character data) available to return in <i>rgbValue</i> prior to the current call to <b>SQLGetData</b>, or SQL_NO_TOTAL if the number of available bytes cannot be determined.</p> <p>For character data, if <i>pcbValue</i> is SQL_NO_TOTAL or is greater than or equal to <i>cbValueMax</i>, the data in <i>rgbValue</i> is truncated to <i>cbValueMax</i> - 1 bytes and is null-terminated by the driver.</p> <p>For binary data, if <i>pcbValue</i> is SQL_NO_TOTAL or is greater than <i>cbValueMax</i>, the data in <i>rgbValue</i> is truncated to <i>cbValueMax</i> bytes.</p> <p>For all other data types, the value of <i>cbValueMax</i> is ignored, and the driver assumes the size of <i>rgbValue</i> is the size of the C data type specified with <i>fCType</i>.</p>

(3 of 3)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NO\_DATA\_FOUND, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetData** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetData** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	All the data for the specified column, <i>icol</i> , could not be retrieved in a single call to the function. The argument <i>pcbValue</i> contains the length of the data that remains in the specified column prior to the current call to <b>SQLGetData</b> (function returns <code>SQL_SUCCESS_WITH_INFO</code> ). For more information on using multiple calls to <b>SQLGetData</b> for a single column, see <a href="#">“Usage” on page 13-171</a> .
07006	Restricted data-type attribute violation	The data value cannot be converted to the C data type specified by the argument <i>fCType</i> .
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
22002	Indicator value required but not supplied	If the column value is null, then <i>StrLen_or_Ind</i> must not be a null pointer.

(1 of 5)

SQLSTATE	Error	Description
22003	Numeric value out of range	<p>Returning the numeric value (as numeric or string) for the column would have caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>Returning the binary value for the column would have caused a loss of binary significance.</p> <p>For more information, see <a href="#">Appendix C</a>.</p>
22005	Error in assignment	The data for the column was incompatible with the data type into which it was to be converted. For more information, see <a href="#">Appendix C</a> .
22008	Datetime-field overflow	The data for the column was not a valid date, time, or time-stamp value. For more information, see <a href="#">Appendix C</a> .
22012	Division by zero	A value from an arithmetic expression was returned that resulted in a division by zero.
24000	Invalid cursor state	<p>(DM) The <i>hstmt</i> was in an executed state, but no result set was associated with the <i>hstmt</i>.</p> <p>(DM) A cursor was open on the <i>hstmt</i>, but <b>SQLFetch</b> or <b>SQLExtendedFetch</b> had not been called.</p> <p>A cursor was open on the <i>hstmt</i> and <b>SQLFetch</b> or <b>SQLExtendedFetch</b> had been called, but the cursor was positioned before the start of the result set or after the end of the result set.</p>
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.

(2 of 5)

SQLSTATE	Error	Description
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1002	Invalid column number	<p>The value specified for the argument <i>icol</i> was 0, and <b>SQLFetch</b> was used to fetch the data.</p> <p>The value specified for the argument <i>icol</i> was 0, and the <b>SQL_USE_BOOKMARKS</b> statement option was set to <b>SQL_UB_OFF</b>.</p> <p>The specified column was greater than the number of result columns.</p> <p>The specified column was bound through a call to <b>SQLBindCol</b>. This description does not apply to drivers that return the <b>SQL_GD_BOUND</b> bitmask for the <b>SQL_GETDATA_EXTENSIONS</b> option in <b>SQLGetInfo</b>.</p> <p>The specified column was at or before the last bound column specified through <b>SQLBindCol</b>. This description does not apply to drivers that return the <b>SQL_GD_ANY_COLUMN</b> bitmask for the <b>SQL_GETDATA_EXTENSIONS</b> option in <b>SQLGetInfo</b>.</p> <p>The application already called <b>SQLGetData</b> for the current row. The column specified in the current call was before the column specified in the preceding call. This description does not apply to drivers that return the <b>SQL_GD_ANY_ORDER</b> bitmask for the <b>SQL_GETDATA_EXTENSIONS</b> option in <b>SQLGetInfo</b>.</p>
S1003	Program type out of range	(DM) The argument <i>fCType</i> was not a valid data type or <b>SQL_C_DEFAULT</b> .

(3 of 5)

SQLSTATE	Error	Description
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1009	Invalid argument value	(DM) The argument <i>rgbValue</i> was a null pointer.
S1010	Function-sequence error	(DM) The specified <i>hstmt</i> was not in an executed state. The function was called without first calling <b>SQLExecDirect</b> , <b>SQLExecute</b> , or a catalog function.  (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbValueMax</i> was less than 0.
S1109	Invalid cursor position	The cursor was positioned (by <b>SQLExtendedFetch</b> ) on a row for which the value in the <i>rgfRowStatus</i> array in <b>SQLExtendedFetch</b> was SQL_ROW_DELETED or SQL_ROW_ERROR.

(4 of 5)

SQLSTATE	Error	Description
S1C00	Driver not capable	<p>The driver or data source does not support the use of <b>SQLGetData</b> with multiple rows in <b>SQLExtendedFetch</b>. This description does not apply to drivers that return the <code>SQL_GD_BLOCK</code> bitmask for the <code>SQL_GETDATA_EXTENSIONS</code> option in <b>SQLGetInfo</b>.</p> <p>The driver or data source does not support the conversion specified by the combination of the <i>fCType</i> argument and the SQL data type of the corresponding column. This error applies only when the SQL data type of the column was mapped to a driver-specific SQL data type.</p> <p>The argument <i>icol</i> was 0, and the driver does not support bookmarks.</p>
S1T00	Time-out expired	<p>The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b>, <code>SQL_QUERY_TIMEOUT</code>.</p>

(5 of 5)

## Usage

With each call, the driver sets *pcbValue* to the number of bytes that are available in the result column before the current call to **SQLGetData**. If **SQLSetStmtOption** sets `SQL_MAX_LENGTH` and the total number of bytes that are available on the first call is greater than `SQL_MAX_LENGTH`, the available number of bytes is set to `SQL_MAX_LENGTH`.

The `SQL_MAX_LENGTH` statement option is intended to reduce network traffic and might not be supported by all drivers. To guarantee that data is truncated, an application should allocate a buffer of the desired size and specify this size in the *cbValueMax* argument. If the total number of bytes that are in the result column cannot be determined in advance, the driver sets *pcbValue* to `SQL_NO_TOTAL`. If the data value for the column is `NULL`, the driver stores `SQL_NULL_DATA` in *pcbValue*.

**SQLGetData** can convert data to a different data type. The result and success of the conversion is determined by the rules for assignment specified in “Converting Data from SQL to C Data Types” in Appendix D, “Data Types.”

If the application requires more than one call to **SQLGetData** to retrieve data from a single column with a character, binary, or data source-specific data type, the driver returns `SQL_SUCCESS_WITH_INFO`. A subsequent call to **SQLError** returns `SQLSTATE 01004` (Data truncated). The application can then use the same column number to retrieve subsequent parts of the data until **SQLGetData** returns `SQL_SUCCESS`, indicating that all the column data has been retrieved. **SQLGetData** returns `SQL_NO_DATA_FOUND` when it calls for a column after all the data has been retrieved and before data is retrieved for a subsequent column. The application can ignore excess data by proceeding to the next result column.



**Important:** An application can use **SQLGetData** to retrieve data from a column in parts only when it retrieves character C data from a column with a character, binary, or data-source-specific data type or when it retrieves binary C data from a column with a character, binary, or data-source-specific data type. If **SQLGetData** is called more than once in a row for a column under any other conditions, it returns `SQL_NO_DATA_FOUND` for all calls after the first call.

For maximum interoperability, applications should call **SQLGetData** only for unbound columns with numbers greater than the number of the last bound column. Within a single row of data, the column number in each call to **SQLGetData** should be greater than or equal to the column number in the previous call (that is, data should be retrieved in increasing order of column number). As extended functionality, drivers can return data through **SQLGetData** from bound columns, from columns before the last bound column, or from columns in any order. To determine whether a driver supports these extensions, an application calls **SQLGetInfo** with the `SQL_GETDATA_EXTENSIONS` option.

Furthermore, applications that use **SQLExtendedFetch** to retrieve data should call **SQLGetData** only when the rowset size is 1.



## Code Example

In the following example, an application executes a `SELECT` statement to return a result set of the employee names, ages, and birthdays sorted by birthday, age, and name. For each row of data, it calls `SQLFetch` to position the cursor to the next row. It calls `SQLGetData` to retrieve the fetched data; the storage locations for the data and the returned number of bytes are specified in the call to `SQLGetData`. Finally, it prints each employee's name, age, and birthday.

```
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR    szName[NAME_LEN], szBirthday[BDAY_LEN];
SWORD    sAge;
SDWORD   cbName, cbAge, cbBirthday;

retcode = SQLExecDirect(hstmt,

    "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE ORDER BY 3, 2, 1",
    SQL_NTS);

if (retcode == SQL_SUCCESS) {
    while (TRUE) {
        retcode = SQLFetch(hstmt);
        if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
            show_error();
        }
        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
            /* Get data for columns 1, 2, and 3 */
            /* Print the row of data          */

            SQLGetData(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
            SQLGetData(hstmt, 2, SQL_C_SSHORT, &sAge, 0, &cbAge);
            SQLGetData(hstmt, 3, SQL_C_CHAR, szBirthday, BDAY_LEN,

                &cbBirthday);

            fprintf(out, "%-*s %-2d %*s", NAME_LEN-1, szName, sAge,

                BDAY_LEN-1, szBirthday);
        } else {
            break;
        }
    }
}
```

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Sending parameter data at execution time	<b>SQLPutData</b>

## SQLGetFunctions

### Level 1

**SQLGetFunctions** returns information about whether a driver supports a specific ODBC function. This function is implemented in the driver manager; it can also be implemented in drivers. If a driver implements **SQLGetFunctions**, the driver manager calls the function in the driver. Otherwise, it executes the function.

## Syntax

```
RETCODE SQLGetFunctions(hdbc, fFunction, pfExists)
```

The **SQLGetFunctions** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
UWORD	<i>fFunction</i>	Input	SQL_API_ALL_FUNCTIONS or a <b>#define</b> value that identifies the ODBC function of interest. For a list of <b>#define</b> values that identify ODBC functions, see “Usage” on page 13-177.
UWORD FAR *	<i>pfExists</i>	Output	<p>If <i>fFunction</i> is SQL_API_ALL_FUNCTIONS, <i>pfExists</i> points to a UWORD array with 100 elements. The array is indexed by <b>#define</b> values used by <i>fFunction</i> to identify each ODBC function; some elements of the array are unused and are reserved for future use. An element is TRUE if it identifies an ODBC function supported by the driver. It is FALSE if it identifies an ODBC function not supported by the driver or does not identify an ODBC function.</p> <p>The <i>fFunction</i> value SQL_API_ALL_FUNCTIONS was added in ODBC 2.0.</p> <p>If <i>fFunction</i> identifies a single ODBC function, <i>pfExists</i> points to a single UWORD. <i>pfExists</i> is TRUE if the specified function is supported by the driver; otherwise, it is FALSE.</p>

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetFunctions** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetFunctions** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) <b>SQLGetFunctions</b> was called before <b>SQLConnect</b> , <b>SQLBrowseConnect</b> , or <b>SQLDriverConnect</b> .  (DM) <b>SQLBrowseConnect</b> was called for the <i>hdbc</i> and returned <code>SQL_NEED_DATA</code> . This function was called before <b>SQLBrowseConnect</b> returned <code>SQL_SUCCESS_WITH_INFO</code> or <code>SQL_SUCCESS</code> .
S1095	Function type out of range	(DM) An invalid <i>fFunction</i> value was specified.

## Usage

**SQLGetFunctions** always returns that **SQLGetFunctions**, **SQLDataSources**, and **SQLDrivers** are supported. It does so because these functions are implemented in the driver manager.

The following list identifies valid values for *fFunction* for ODBC core functions that the INFORMIX-CLI driver supports:

- SQL\_API\_SQLALLOCONNECT
- SQL\_API\_SQLALLOCENV
- SQL\_API\_SQLALLOCSTMT
- SQL\_API\_SQLBINDCOL
- SQL\_API\_SQLCANCEL
- SQL\_API\_SQLCOLATTRIBUTES
- SQL\_API\_SQLCONNECT
- SQL\_API\_SQLDESCRIBECOL
- SQL\_API\_SQLDISCONNECT
- SQL\_API\_SQLERROR
- SQL\_API\_SQLEXECDIRECT
- SQL\_API\_SQLEXECUTE
- SQL\_API\_SQLFETCH
- SQL\_API\_SQLFREECONNECT
- SQL\_API\_SQLFREEENV
- SQL\_API\_SQLFREESTMT
- SQL\_API\_SQLGETCURSORNAME
- SQL\_API\_SQLNUMRESULTCOLS
- SQL\_API\_SQLPREPARE
- SQL\_API\_SQLROWCOUNT
- SQL\_API\_SQLSETCURSORNAME
- SQL\_API\_SQLSETPARAM
- SQL\_API\_SQLTRANSACT

The following list identifies valid values for *fFunction* for ODBC extension level 1 functions that the INFORMIX-CLI driver supports:

- SQL\_API\_SQLBINDPARAMETER
- SQL\_API\_SQLCOLUMNS
- SQL\_API\_SQLDRIVERCONNECT
- SQL\_API\_SQLGETCONNECTOPTION
- SQL\_API\_SQLGETDATA
- SQL\_API\_SQLGETFUNCTIONS
- SQL\_API\_SQLGETINFO
- SQL\_API\_SQLGETSTMTOPTION
- SQL\_API\_SQLGETTYPEINFO
- SQL\_API\_SQLPARAMDATA
- SQL\_API\_SQLPUTDATA
- SQL\_API\_SQLSETCONNECTOPTION
- SQL\_API\_SQLSETSTMTOPTION
- SQL\_API\_SQLSPECIALCOLUMNS
- SQL\_API\_SQLSTATISTICS
- SQL\_API\_SQLTABLES

The following list identifies valid values for *fFunction* for ODBC extension level 2 functions that the INFORMIX-CLI driver supports:

- SQL\_API\_SQLBROWSECONNECT
- SQL\_API\_COLUMNPRIVILEGES
- SQL\_API\_SQLDATASOURCES
- SQL\_API\_SQLDRIVERS
- SQL\_API\_SQLEXTENDEDFETCH
- SQL\_API\_MORERESULTS
- SQL\_API\_SQLNATIVESQL
- SQL\_API\_SQLNUMPARAMS
- SQL\_API\_PARAMOPTIONS
- SQL\_API\_SQLPRIMARYKEYS
- SQL\_API\_PROCEDURES
- SQL\_API\_SQLSETSCROLLOPTIONS
- SQL\_API\_TABLEPRIVILEGES

## Code Example

The following examples show how an application uses **SQLGetFunctions** to determine if a driver supports **SQLTables**, **SQLColumns**, and **SQLStatistics**. If the driver does not support these functions, the application disconnects from the driver. The first example calls **SQLGetFunctions** once for each function.

```

UWORD TablesExists, ColumnsExists, StatisticsExists;

SQLGetFunctions(hdbc, SQL_API_SQLTABLES, &TablesExists);
SQLGetFunctions(hdbc, SQL_API_SQLCOLUMNS, &ColumnsExists);
SQLGetFunctions(hdbc, SQL_API_SQLSTATISTICS,
&StatisticsExists);

if (TablesExists && ColumnsExists && StatisticsExists) {

    /* Continue with application */

}

SQLDisconnect(hdbc);

```

The second example calls **SQLGetFunctions** once and passes it an array in which **SQLGetFunctions** returns information about all INFORMIX-CLI functions.

```

UWORD fExists[100];

SQLGetFunctions(hdbc, SQL_API_ALL_FUNCTIONS, fExists);

if (fExists[SQL_API_SQLTABLES] &&
    fExists[SQL_API_SQLCOLUMNS] &&
    fExists[SQL_API_SQLSTATISTICS]) {

    /* Continue with application */

}

SQLDisconnect(hdbc);

```

## Related Functions

<b>For information about</b>	<b>See</b>
Returning the setting of a connection option	<b>SQLGetConnectOption</b>
Returning information about a driver or data source	<b>SQLGetInfo</b>
Returning the setting of a statement option	<b>SQLGetStmtOption</b>



## SQLGetInfo

**SQLGetInfo** returns general information about the driver and data source associated with an *hdbc*.

### Syntax

```
RETCODE SQLGetInfo(hdbc, fInfoType, rgbInfoValue,
                  cbInfoValueMax, pcbInfoValue)
```

The **SQLGetInfo** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
UWORD	<i>fInfoType</i>	Input	Type of information.

The *fInfoType* argument must be a value that represents the type of interest (see [“Usage” on page 13-184](#)).

(1 of 2)

Type	Argument	Use	Description
PTR	<i>rgbInfoValue</i>	Output	Pointer to storage for the information. Depending on the <i>fInfoType</i> requested, the information returned will be one of the following: a null-terminated character string, a 16-bit integer value, a 32-bit flag, or a 32-bit binary value.
SWORD	<i>cbInfoValueMax</i>	Input	Maximum length of the <i>rgbInfoValue</i> buffer.
SWORD FAR *	<i>pcbInfoValue</i>	Output	<p>The total number of bytes (excluding the null-termination byte for character data) available to return in <i>rgbInfoValue</i>.</p> <p>For character data, if the number of bytes available to return is greater than or equal to <i>cbInfoValueMax</i>, the information in <i>rgbInfoValue</i> is truncated to <i>cbInfoValueMax</i> - 1 bytes and is null-terminated by the driver.</p> <p>For all other types of data, the value of <i>cbValueMax</i> is ignored, and the driver assumes the size of <i>rgbValue</i> is 32 bits.</p>

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetInfo** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetInfo** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The buffer <i>rgbInfoValue</i> was not large enough to return all of the requested information, so the information was truncated. The argument <i>pcbInfoValue</i> contains the length of the requested information in its untruncated form (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
08003	Connection not open	(DM) The type of information requested in <i>fInfoType</i> requires an open connection. Of the information types reserved by ODBC, only <code>SQL_ODBC_VER</code> can be returned without an open connection.
22003	Numeric value out of range	Returning the requested information would have caused a loss of numeric or binary significance.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

(1 of 2)

SQLSTATE	Error	Description
S1009	Invalid argument value	(DM) The <i>fInfoType</i> was SQL_DRIVER_HSTMT, and the value pointed to by <i>rgbInfoValue</i> was not a valid statement handle.
S1090	Invalid string or buffer length	(DM) The value specified for argument <i>cbInfoValueMax</i> was less than 0.
S1096	Information type out of range	(DM) The value specified for the argument <i>fOption</i> was in the block of numbers reserved for ODBC information types but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The driver does not support the value specified for the argument <i>fOption</i> .
S1T00	Time-out expired	The time-out period expired before the data source returned the requested information. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

## Usage

The format of the information returned in *rgbInfoValue* depends on the *fInfoType* requested. **SQLGetInfo** returns information in one of five formats:

- A null-terminated character string
- A 16-bit integer value
- A 32-bit bitmask
- A 32-bit integer value
- A 32-bit binary value

The format for each of the following information types is noted in the description. The application must cast the value returned in *rgbInfoValue* accordingly. For an example of how an application could retrieve data from a 32-bit bitmask, see [“Code Example” on page 13-211](#).

A driver must return a value for each information type defined in the following tables. If an information type does not apply to the driver or data source, the driver returns one of the following values.

Format of <i>rgbInfoValue</i>	Returned value
Character string (“Y” or “N”)	N
Character string (not “Y” or “N”)	Empty string
16-bit integer	0
32-bit bitmask or 32-bit binary value	0L

For example, if a data source does not support procedures, **SQLGetInfo** returns the following values for the *fInfoType* values, which are related to procedures.

<i>fInfoType</i>	Returned value
SQL_PROCEDURES	N
SQL_ACCESSIBLE_PROCEDURES	N
SQL_MAX_PROCEDURE_NAME_LEN	0
SQL_PROCEDURE_TERM	Empty string

**SQLGetInfo** returns SQLSTATE S1096 (Invalid argument value) for a value of *fInfoType* that is not defined by the version of ODBC the driver supports. To determine the version of ODBC to which a driver conforms, an application calls **SQLGetInfo** with the SQL\_DRIVER\_ODBC\_VER information type. **SQLGetInfo** returns SQLSTATE S1C00 (Driver not capable) for values of *fInfoType* that are in the range of information types reserved for driver-specific use but that the driver does not support.

### ***Information Types***

This section lists the information types supported by **SQLGetInfo**. Information types are grouped categorically and listed alphabetically.

### *Driver Information*

The following values of *fnfoType* return information about the Informix driver, such as the number of active statements, the data source name, and the API conformance levels:

- SQL\_ACTIVE\_CONNECTIONS
- SQL\_ACTIVE\_STATEMENTS
- SQL\_DATA\_SOURCE\_NAME
- SQL\_DRIVER\_HDBC
- SQL\_DRIVER\_HENV
- SQL\_DRIVER\_HLIB
- SQL\_DRIVER\_HSTMT
- SQL\_DRIVER\_NAME
- SQL\_DRIVER\_ODBC\_VER
- SQL\_DRIVER\_VER
- SQL\_FETCH\_DIRECTION
- SQL\_FILE\_USAGE
- SQL\_GETDATA\_EXTENSIONS
- SQL\_LOCK\_TYPES
- SQL\_ODBC\_API\_CONFORMANCE
- SQL\_ODBC\_SAG\_CLI\_CONFORMANCE
- SQL\_ODBC\_VER
- SQL\_POS\_OPERATIONS
- SQL\_ROW\_UPDATES
- SQL\_SEARCH\_PATTERN\_ESCAPE
- SQL\_SERVER\_NAME

### *DBMS Product Information*

The following values of *fnfoType* return information about the Informix DBMS product, such as the DBMS name and version:

- SQL\_DATABASE\_NAME
- SQL\_DBMS\_NAME
- SQL\_DBMS\_VER

*Data-Source Information*

The following values of *flInfoType* return information about the data source, such as cursor characteristics and transaction capabilities:

- SQL\_ACCESSIBLE\_PROCEDURES
- SQL\_ACCESSIBLE\_TABLES
- SQL\_CONCAT\_NULL\_BEHAVIOR
- SQL\_CURSOR\_COMMIT\_BEHAVIOR
- SQL\_CURSOR\_ROLLBACK\_BEHAVIOR
- SQL\_DATA\_SOURCE\_READ\_ONLY
- SQL\_DEFAULT\_TXN\_ISOLATION
- SQL\_MULT\_RESULT\_SETS
- SQL\_MULTIPLE\_ACTIVE\_TXN
- SQL\_NEED\_LONG\_DATA\_LEN
- SQL\_NULL\_COLLATION
- SQL\_OWNER\_TERM
- SQL\_PROCEDURE\_TERM
- SQL\_QUALIFIER\_TERM
- SQL\_SCROLL\_CONCURRENCY
- SQL\_SCROLL\_OPTIONS
- SQL\_STATIC\_SENSITIVITY
- SQL\_TABLE\_TERM
- SQL\_TXN\_CAPABLE
- SQL\_TXN\_ISOLATION\_OPTION
- SQL\_USER\_NAME

### *Supported SQL*

The following values of *flInfoType* return information about the SQL statements that are supported by the data source. These information types do not exhaustively describe ODBC SQL grammar, but they describe those parts of the grammar for which data sources commonly offer different levels of support.

Applications should determine the general level of supported grammar from the SQL\_ODBC\_SQL\_CONFORMANCE information type and use the other information types to determine variations from the stated conformance level:

- SQL\_ALTER\_TABLE
- SQL\_COLUMN\_ALIAS
- SQL\_CORRELATION\_NAME
- SQL\_EXPRESSIONS\_IN\_ORDERBY
- SQL\_GROUP\_BY
- SQL\_IDENTIFIER\_CASE
- SQL\_IDENTIFIER\_QUOTE\_CHAR
- SQL\_KEYWORDS
- SQL\_LIKE\_ESCAPE\_CLAUSE
- SQL\_NON\_NULLABLE\_COLUMNS
- SQL\_ODBC\_SQL\_CONFORMANCE
- SQL\_ODBC\_SQL\_OPT\_IEF
- SQL\_ORDER\_BY\_COLUMNS\_IN\_SELECT
- SQL\_OUTER\_JOINS
- SQL\_OWNER\_USAGE
- SQL\_POSITIONED\_STATEMENTS
- SQL\_PROCEDURES
- SQL\_QUALIFIER\_LOCATION
- SQL\_QUALIFIER\_NAME\_SEPARATOR
- SQL\_QUALIFIER\_USAGE
- SQL\_QUOTED\_IDENTIFIER\_CASE
- SQL\_SPECIAL\_CHARACTERS
- SQL\_SUBQUERIES
- SQL\_UNION



### SQL Limits

The following values of *fnfoType* return information about the limits applied to identifiers and clauses in SQL statements, such as the maximum lengths of identifiers and the maximum number of columns in a SELECT list. Limitations might be imposed by either the driver or the data source:

- SQL\_MAX\_BINARY\_LITERAL\_LEN
- SQL\_MAX\_CHAR\_LITERAL\_LEN
- SQL\_MAX\_COLUMN\_NAME\_LEN
- SQL\_MAX\_COLUMNS\_IN\_GROUP\_BY
- SQL\_MAX\_COLUMNS\_IN\_ORDER\_BY
- SQL\_MAX\_COLUMNS\_IN\_INDEX
- SQL\_MAX\_COLUMNS\_IN\_SELECT
- SQL\_MAX\_COLUMNS\_IN\_TABLE
- SQL\_MAX\_CURSOR\_NAME\_LEN
- SQL\_MAX\_INDEX\_SIZE
- SQL\_MAX\_OWNER\_NAME\_LEN
- SQL\_MAX\_PROCEDURE\_NAME\_LEN
- SQL\_MAX\_QUALIFIER\_NAME\_LEN
- SQL\_MAX\_ROW\_SIZE
- SQL\_MAX\_ROW\_SIZE\_INCLUDES\_LONG
- SQL\_MAX\_STATEMENT\_LEN
- SQL\_MAX\_TABLE\_NAME\_LEN
- SQL\_MAX\_TABLES\_IN\_SELECT
- SQL\_MAX\_USER\_NAME\_LEN

*Scalar Function Information*

The following values of *InfoType* return information about the scalar functions supported by the data source and the driver: For more information on scalar functions, refer to the [Informix Guide to SQL: Syntax](#).

- SQL\_NUMERIC\_FUNCTIONS
- SQL\_STRING\_FUNCTIONS
- SQL\_SYSTEM\_FUNCTIONS
- SQL\_TIMEDATE\_ADD\_INTERVALS
- SQL\_TIMEDATE\_DIFF\_INTERVALS
- SQL\_TIMEDATE\_FUNCTIONS

*Information-Type Descriptions*

The following table alphabetically lists each information type and its description.

InfoType	Returns
SQL_ACCESSIBLE_PROCEDURES	A character string: “Y” if the user can execute all procedures returned by <b>SQLProcedures</b> ; “N” if procedures might be returned that the user cannot execute.
SQL_ACCESSIBLE_TABLES	A character string: “Y” if the user is guaranteed SELECT privileges to all tables returned by <b>SQLTables</b> , “N” if tables might be returned that the user cannot access.
SQL_ACTIVE_CONNECTIONS	A 16-bit integer value that specifies the maximum number of active <i>hdbcs</i> that the driver can support. This value can reflect a limitation imposed by either the driver or the data source. If no limit is specified or the limit is unknown, this value is set to 0.
SQL_ACTIVE_STATEMENTS	A 16-bit integer value that specifies the maximum number of active <i>hstmts</i> that the driver can support for an <i>hdbc</i> . This value can reflect a limitation imposed by either the driver or the data source. If no limit is specified or the limit is unknown, this value is set to 0.

(1 of 22)

InfoType	Returns
SQL_ALTER_TABLE	<p>A 32-bit bitmask that enumerates the clauses in the ALTER TABLE statement supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_AT_ADD_COLUMN SQL_AT_DROP_COLUMN</p>
SQL_BOOKMARK_PERSISTENCE UNSUPPORTED	INFORMIX-CLI does not support this option.
SQL_COLUMN_ALIAS	A character string: “Y” if the data source supports column aliases; otherwise, “N” is returned.
SQL_CONCAT_NULL_BEHAVIOR	<p>A 16-bit integer value that indicates how the data source handles the concatenation of NULL-valued character-data-type columns with non-NULL-valued character-data-type columns:</p> <p>SQL_CB_NULL = Result is NULL valued.</p> <p>SQL_CB_NON_NULL = Result is concatenation of non-NULL-valued column or columns.</p>
SQL_CONVERT_FUNCTIONS UNSUPPORTED	INFORMIX-CLI does not support this option.
SQL_CORRELATION_NAME	<p>A 16-bit integer that indicates if table correlation names are supported:</p> <p>SQL_CN_NONE = Correlation names are not supported.</p> <p>SQL_CN_DIFFERENT = Correlation names are supported, but they must differ from the names of the tables that they represent.</p> <p>SQL_CN_ANY = Correlation names are supported and can be any valid user-defined name.</p>

(2 of 22)

InfoType	Returns
SQL_CURSOR_COMMIT_BEHAVIOR	<p>A 16-bit integer value that indicates how a COMMIT operation affects cursors and prepared statements in the data source:</p> <p>SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must reprepare and reexecute the <i>hstmt</i>.</p> <p>SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call <b>SQLExecute</b> on the <i>hstmt</i> without calling <b>SQLPrepare</b> again.</p> <p>SQL_CB_PRESERVE = Preserve cursors in the same position as before the COMMIT operation. The application can continue to fetch data or it can close the cursor and reexecute the <i>hstmt</i> without repreparing it.</p>
SQL_CURSOR_ROLLBACK_BEHAVIOR	<p>A 16-bit integer value that indicates how a ROLLBACK operation affects cursors and prepared statements in the data source:</p> <p>SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must reprepare and reexecute the <i>hstmt</i>.</p> <p>SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call <b>SQLExecute</b> on the <i>hstmt</i> without calling <b>SQLPrepare</b> again.</p> <p>SQL_CB_PRESERVE = Preserve cursors in the same position as before the ROLLBACK operation. The application can continue to fetch data, or it can close the cursor and reexecute the <i>hstmt</i> without repreparing it.</p>
SQL_DATA_SOURCE_NAME	<p>A character string with the data-source name used during connection. If the application called <b>SQLConnect</b>, this is the value of the <i>szDSN</i> argument. If the application called <b>SQLDriverConnect</b> or <b>SQLBrowseConnect</b>, this 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 (as when it contains the DRIVER keyword), this is an empty string.</p>
SQL_DATA_SOURCE_READ_ONLY	<p>A character string: "Y" if the data source is set to READ ONLY mode, "N" if it is otherwise.</p> <p>This characteristic pertains only to the data source; it is not a characteristic of the driver that enables access to the data source.</p>

InfoType	Returns
SQL_DATABASE_NAME	A character string with the name of the current database in use, if the data source defines a named object called “database.” In ODBC 2.0, this value of <i>InfoType</i> has been replaced by the SQL_CURRENT_QUALIFIER connection option.
SQL_DBMS_NAME	A character string with the name of the DBMS product accessed by the driver.
SQL_DBMS_VER	A character string that indicates the version of the DBMS product accessed by the driver. The version has the form ##.##.####, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The driver must render the DBMS product version in this form but can also append the DBMS product-specific version. For example, “04.01.0000 Rdb 4.1.”

(4 of 22)

InfoType	Returns
SQL_DEFAULT_TXN_ISOLATION	<p>A 32-bit integer that indicates the default transaction isolation level supported by the driver or data source, or zero if the data source does not support transactions. The following terms are used to define transaction isolation levels:</p> <p><i>Dirty Read</i> Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed.</p> <p><i>Nonrepeatable Read</i> Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted.</p> <p><i>Phantom</i> Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 reexecutes the statement that read the rows, it receives a different set of rows.</p> <p>If the data source supports transactions, the driver returns one of the following bitmasks:</p> <p>SQL_TXN_READ_UNCOMMITTED = Dirty Reads, Nonrepeatable Reads, and Phantoms are possible.</p> <p>SQL_TXN_READ_COMMITTED = Dirty Reads are not possible. Nonrepeatable Reads and Phantoms are possible.</p> <p>SQL_TXN_REPEATABLE_READ = Dirty Reads and Nonrepeatable Reads are not possible. Phantoms are possible.</p> <p>SQL_TXN_SERIALIZABLE = Transactions are serializable. Dirty Reads, Nonrepeatable Reads, and Phantoms are not possible.</p> <p>SQL_TXN_VERSIONING = Transactions are serializable, but higher concurrency is possible than with SQL_TXN_SERIALIZABLE. Dirty Reads are not possible. Typically, SQL_TXN_SERIALIZABLE is implemented by using locking protocols that reduce concurrency, and SQL_TXN_VERSIONING is implemented by using a nonlocking protocol such as record versioning.</p>
SQL_DRIVER_HDBC SQL_DRIVER_HENV	<p>A 32-bit value, the environment handle or connection handle of the driver, determined by the argument <i>hdbc</i>.</p> <p>These information types are implemented by the driver manager alone.</p>

InfoType	Returns
SQL_DRIVER_HLIB	<p>A 32-bit value, the library handle returned to the driver manager when it loaded the driver shared library. The handle is only valid for the <i>hdbc</i> specified in the call to <b>SQLGetInfo</b>.</p> <p>This information type is implemented by the driver manager alone.</p>
SQL_DRIVER_HSTMT	<p>A 32-bit value, the statement handle of the driver determined by the driver manager statement handle, which must be passed on input in <i>rgbInfoValue</i> from the application.</p> <p>In this case, <i>rgbInfoValue</i> is both an input and an output argument. The input <i>hstmt</i> passed in <i>rgbInfoValue</i> was probably an <i>hstmt</i> allocated on the argument <i>hdbc</i>.</p> <p>This information type is implemented by the driver manager alone.</p>
SQL_DRIVER_NAME	<p>A character string with the filename of the driver used to access the data source.</p>
SQL_DRIVER_ODBC_VER	<p>A character string with the version of ODBC that the driver supports. The version has the form <i>##.##</i>, where the first two digits are the major version and the next two digits are the minor version. <i>SQL_SPEC_MAJOR</i> and <i>SQL_SPEC_MINOR</i> define the major and minor version numbers. For the version of ODBC described in this manual, these are 2 and 0, and the driver should return <i>02.00</i>.</p> <p>If a driver supports <b>SQLGetInfo</b> but does not support this value of the <i>fInfoType</i> argument, the driver manager returns <i>01.00</i>.</p>
SQL_DRIVER_VER	<p>A character string with the version of the driver and, optionally, a description of the driver. At a minimum, the version has the form <i>##.##.####</i>, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version.</p>
SQL_EXPRESSIONS_IN_ORDERBY	<p>A character string: “Y” if the data source supports expressions in the ORDER BY list; “N” if it does not.</p>

(6 of 22)

InfoType	Returns
SQL_FETCH_DIRECTION	<p>A 32-bit bitmask that enumerates the supported fetch direction options.</p> <p>The following bitmasks are used in conjunction with the flag to determine which options are supported:</p> <p>SQL_FD_FETCH_NEXT  SQL_FD_FETCH_FIRST)  SQL_FD_FETCH_LAST  SQL_FD_FETCH_PRIOR  SQL_FD_FETCH_ABSOLUTE  SQL_FD_FETCH_RELATIVE  SQL_FD_FETCH_RESUME</p>
SQL_FILE_USAGE	<p>A 16-bit integer value that indicates how a single-tier driver directly treats files in a data source:</p> <p>SQL_FILE_NOT_SUPPORTED = The driver is not a single-tier driver.</p> <p>SQL_FILE_TABLE = A single-tier driver treats files in a data source as tables. For example, a text driver treats each text file as a table.</p> <p>SQL_FILE_QUALIFIER = A single-tier driver treats files in a data source as a qualifier.</p>

(7 of 22)



InfoType	Returns
SQL_GETDATA_EXTENSIONS	<p>A 32-bit bitmask that enumerates extensions to <b>SQLGetData</b>.</p> <p>The following bitmasks are used in conjunction with the flag to determine what common extensions the driver supports for <b>SQLGetData</b>:</p> <p><b>SQL_GD_ANY_COLUMN</b> = <b>SQLGetData</b> can be called for any unbound column, including those before the last bound column. The columns must be called in order of ascending column number unless <b>SQL_GD_ANY_ORDER</b> is also returned.</p> <p><b>SQL_GD_ANY_ORDER</b> = <b>SQLGetData</b> can be called for unbound columns in any order. <b>SQLGetData</b> can be called only for columns after the last bound column unless <b>SQL_GD_ANY_COLUMN</b> is also returned.</p> <p><b>SQL_GD_BOUND</b> = <b>SQLGetData</b> can be called for bound columns as well as unbound columns. A driver cannot return this value unless it also returns <b>SQL_GD_ANY_COLUMN</b>.</p> <p><b>SQLGetData</b> is required to return data only from unbound columns that occur after the last bound column, are called in order of increasing column number, and are not in a row in a block of rows.</p>

(8 of 22)

InfoType	Returns
SQL_GROUP_BY	<p>A 16-bit integer value that specifies the relationship between the columns in the GROUP BY clause and the non-aggregated columns in the SELECT list:</p> <p>SQL_GB_NOT_SUPPORTED = GROUP BY clauses are not supported.</p> <p>SQL_GB_GROUP_BY_EQUALS_SELECT = The GROUP BY clause must contain all nonaggregated columns in the SELECT list. It cannot contain any other columns. For example:</p> <pre data-bbox="583 513 1197 558">SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT</pre> <p>SQL_GB_GROUP_BY_CONTAINS_SELECT = The GROUP BY clause must contain all nonaggregated columns in the SELECT list. It can contain columns that are not in the SELECT list. For example:</p> <pre data-bbox="583 691 1197 737">SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT, AGE</pre> <p>SQL_GB_NO_RELATION = The columns in the GROUP BY clause and the select list are not related. The meaning of nongrouped, nonaggregated columns in the SELECT list is data-source dependent. For example,</p> <pre data-bbox="583 894 1174 940">SELECT DEPT, SALARY FROM EMPLOYEE GROUP BY DEPT, AGE</pre>
SQL_IDENTIFIER_CASE	<p>A 16-bit integer value as follows:</p> <p>SQL_IC_UPPER = Identifiers in SQL are case insensitive and are stored in uppercase in system catalog.</p> <p>SQL_IC_LOWER = Identifiers in SQL are case insensitive and are stored in lowercase in system catalog.</p> <p>SQL_IC_SENSITIVE = Identifiers in SQL are case sensitive and are stored in mixed case in system catalog.</p> <p>SQL_IC_MIXED = Identifiers in SQL are case insensitive and are stored in mixed case in system catalog.</p>
SQL_IDENTIFIER_QUOTE_CHAR	<p>The character string used as the starting and ending delimiter of a quoted (delimited) identifiers in SQL statements. (Identifiers passed as arguments to ODBC functions do not need to be quoted.) If the data source does not support quoted identifiers, a blank is returned.</p>

InfoType	Returns
SQL_KEYWORDS	<p>A character string that contains a comma-separated list of all data-source-specific keywords. This list does not contain keywords specific to ODBC or keywords used by both the data source and ODBC.</p> <p>For a list of ODBC keywords, see <a href="#">“List of Reserved Keywords” on page B-21</a>. The #define value SQL_ODBC_KEYWORDS contains a comma-separated list of ODBC keywords.</p>
SQL_LIKE_ESCAPE_CLAUSE	<p>A character string: “Y” if the data source supports an escape character for the percent character (%) and underscore character (_) in a LIKE predicate and the driver supports the ODBC syntax for defining a LIKE predicate escape character; “N” otherwise.</p>
SQL_MAX_BINARY_LITERAL_LEN	<p>A 32-bit integer value that specifies the maximum length (number of hexadecimal characters, excluding the literal prefix and suffix returned by <b>SQLGetTypeInfo</b>) 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 0.</p>
SQL_MAX_CHAR_LITERAL_LEN	<p>A 32-bit integer value that specifies the maximum length (number of characters, excluding the literal prefix and suffix returned by <b>SQLGetTypeInfo</b>) of a character literal in an SQL statement. If there is no maximum length or the length is unknown, this value is set to 0.</p>
SQL_MAX_COLUMN_NAME_LEN	<p>A 16-bit integer value that specifies the maximum length of a column name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.</p>
SQL_MAX_COLUMNS_IN_GROUP_BY	<p>A 16-bit integer value that specifies the maximum number of columns allowed in a GROUP BY clause. If no limit is specified or the limit is unknown, this value is set to 0.</p>
SQL_MAX_COLUMNS_IN_INDEX	<p>A 16-bit integer value that specifies the maximum number of columns allowed in an index. If no limit is specified or the limit is unknown, this value is set to 0.</p>
SQL_MAX_COLUMNS_IN_ORDER_BY	<p>A 16-bit integer value that specifies the maximum number of columns allowed in an ORDER BY clause. If no limit is specified or the limit is unknown, this value is set to 0.</p>

(10 of 22)

InfoType	Returns
SQL_MAX_COLUMNS_IN_SELECT	A 16-bit integer value that specifies the maximum number of columns allowed in a SELECT list. If no limit is specified or the limit is unknown, this value is set to 0.
SQL_MAX_COLUMNS_IN_TABLE	A 16-bit integer value that specifies the maximum number of columns allowed in a table. If no limit is specified or the limit is unknown, this value is set to 0.
SQL_MAX_CURSOR_NAME_LEN	A 16-bit integer value that specifies the maximum length of a cursor name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.
SQL_MAX_INDEX_SIZE	A 32-bit integer value that specifies the maximum number of bytes allowed in the combined fields of an index. If no limit is specified or the limit is unknown, this value is set to 0.
SQL_MAX_OWNER_NAME_LEN	A 16-bit integer value that specifies the maximum length of an owner name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.
SQL_MAX_PROCEDURE_NAME_LEN	A 16-bit integer value that specifies the maximum length of a procedure name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.
SQL_MAX_QUALIFIER_NAME_LEN	A 16-bit integer value that specifies the maximum length of a qualifier name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.
SQL_MAX_ROW_SIZE	A 32-bit integer value that specifies the maximum length of a single row in a table. If no limit is specified or the limit is unknown, this value is set to 0.
SQL_MAX_ROW_SIZE_INCLUDES_LONG	A character string: "Y" if the maximum row size returned for the SQL_MAX_ROW_SIZE information type includes the length of all SQL_LONGVARCHAR and SQL_LONGVARBINARY columns in the row; "N" otherwise.
SQL_MAX_STATEMENT_LEN	A 32-bit integer value that specifies the maximum length (number of characters, including white space) of an SQL statement. If there is no maximum length or the length is unknown, this value is set to 0.

InfoType	Returns
SQL_MAX_TABLE_NAME_LEN	A 16-bit integer value that specifies the maximum length of a table name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.
SQL_MAX_TABLES_IN_SELECT	A 16-bit integer value that specifies the maximum number of tables allowed in the FROM clause of a SELECT statement. If there is no specified limit or the limit is unknown, this value is set to 0.
SQL_MAX_USER_NAME_LEN	A 16-bit integer value that specifies the maximum length of a user name in the data source. If there is no maximum length or the length is unknown, this value is set to 0.
SQL_MULT_RESULT_SETS	A character string: “Y” if the data source supports multiple result sets, “N” if it does not.
SQL_MULTIPLE_ACTIVE_TXN	A character string: “Y” if active transactions on multiple connections are allowed, “N” if only one connection at a time can have an active transaction.
SQL_NEED_LONG_DATA_LEN	A character string: “Y” if the data source needs the length of a long data value (the data type is SQL_LONGVARCHAR, SQL_LONGVARIABLE, or a long, data-source-specific data type) before that value is sent to the data source, “N” if it does not. For more information, see <b>SQLBindParameter</b> .
SQL_NON_NULLABLE_COLUMNS	A 16-bit integer that specifies whether the data source supports non-nullable columns: SQL_NNC_NULL = All columns must be nullable. SQL_NNC_NON_NULL = Columns may be non-nullable (the data source supports the NOT NULL column constraint in CREATE TABLE statements).
SQL_NULL_COLLATION	A 16-bit integer value that specifies where NULLs are sorted in a list: SQL_NC_END = NULLs are sorted at the end of the list, regardless of the sort order. SQL_NC_HIGH = NULLs are sorted at the high end of the list. SQL_NC_LOW = NULLs are sorted at the low end of the list. SQL_NC_START = NULLs are sorted at the start of the list, regardless of the sort order.

InfoType	Returns
SQL_NUMERIC_FUNCTIONS	<p>A 32-bit bitmask that enumerates the scalar numeric functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which numeric functions are supported:</p> <ul style="list-style-type: none"> <li>SQL_FN_NUM_ABS</li> <li>SQL_FN_NUM_ACOS</li> <li>SQL_FN_NUM_ASIN</li> <li>SQL_FN_NUM_ATAN</li> <li>SQL_FN_NUM_ATAN2</li> <li>SQL_FN_NUM_CEILING</li> <li>SQL_FN_NUM_COS</li> <li>SQL_FN_NUM_COS</li> <li>SQL_FN_NUM_DEGREES</li> <li>SQL_FN_NUM_EXP</li> <li>SQL_FN_NUM_FLOOR</li> <li>SQL_FN_NUM_LOG</li> <li>SQL_FN_NUM_LOG10</li> <li>SQL_FN_NUM_MOD</li> <li>SQL_FN_NUM_PI</li> <li>SQL_FN_NUM_POWER</li> <li>SQL_FN_NUM_RADIANS</li> <li>SQL_FN_NUM_RAND</li> <li>SQL_FN_NUM_ROUND</li> <li>SQL_FN_NUM_SIGN</li> <li>SQL_FN_NUM_SIN</li> <li>SQL_FN_NUM_SQRT</li> <li>SQL_FN_NUM_TAN</li> <li>SQL_FN_NUM_TRUNCATE</li> </ul>
SQL_ODBC_API_CONFORMANCE	<p>A 16-bit integer value that indicates the level of ODBC conformance:</p> <ul style="list-style-type: none"> <li>SQL_OAC_NONE = None</li> <li>SQL_OAC_LEVEL1 = Level 1 supported</li> <li>SQL_OAC_LEVEL2 = Level 2 supported</li> </ul> <p>For a list of functions and conformance levels, see <a href="#">Chapter 12, "Function Summary."</a></p>

InfoType	Returns
SQL_ODBC_SAG_CLI_CONFORMANCE	<p>A 16-bit integer value that indicates compliance to the functions of the SAG specification:</p> <p>SQL_OSCC_NOT_COMPLIANT = Not SAG-compliant; one or more core functions are not supported.</p> <p>SQL_OSCC_COMPLIANT = SAG-compliant</p>
SQL_ODBC_SQL_CONFORMANCE	<p>A 16-bit integer value that indicates SQL grammar supported by the driver:</p> <p>SQL_OSC_MINIMUM = Minimum grammar supported</p> <p>SQL_OSC_CORE = Core grammar supported</p> <p>SQL_OSC_EXTENDED = Extended grammar supported</p>
SQL_ODBC_SQL_OPT_IEF	<p>A character string: “Y” if the data source supports the optional Integrity Enhancement Facility; “N” if it does not.</p>
SQL_ODBC_VER	<p>A character string with the version of ODBC to which the driver manager conforms. The version is of the form ##.##, where the first two digits are the major version and the next two digits are the minor version. This is implemented solely in the driver manager.</p>
SQL_ORDER_BY_COLUMNS_IN_SELECT	<p>A character string: “Y” if the columns in the ORDER BY clause must be in the SELECT list; otherwise, “N.”</p>
SQL_OUTER_JOINS	<p>A character string:</p> <p>“N” = No. The data source does not support outer joins.</p> <p>“Y” = Yes. The data source supports two-table outer joins, and the driver supports the ODBC outer join syntax except for nested outer joins. However, columns on the left side of the comparison operator in the ON clause must come from the left-hand table in the outer join, and columns on the right side of the comparison operator must come from the right-hand table.</p> <p>“P” = Partial. The data source partially supports nested outer joins, and the driver supports the ODBC outer-join syntax. However, columns on the left side of the comparison operator in the ON clause must come from the left-hand table in the outer join and columns on the right side of the comparison operator must come from the right-hand table. Also, the right-hand table of an outer join cannot be included in an inner join.</p> <p>“F” = Full. The data source fully supports nested outer joins, and the driver supports the ODBC outer-join syntax.</p>

InfoType	Returns
SQL_OWNER_TERM	A character string with the data-source vendor name for an owner; for example, "owner," "Authorization ID," or "Schema."
SQL_OWNER_USAGE	<p>A 32-bit bitmask that enumerates the statements in which owners can be used:</p> <p>SQL_OU_DML_STATEMENTS = Owners are supported in all Data Manipulation Language statements: SELECT, INSERT, UPDATE, DELETE, and, if supported, SELECT FOR UPDATE and positioned UPDATE and DELETE statements.</p> <p>SQL_OU_PROCEDURE_INVOCATION = Owners are supported in the ODBC procedure invocation statement.</p> <p>SQL_OU_TABLE_DEFINITION = Owners are supported in all table definition statements: CREATE TABLE, CREATE VIEW, ALTER TABLE, DROP TABLE, and DROP VIEW.</p> <p>SQL_OU_INDEX_DEFINITION = Owners are supported in all index-definition statements: CREATE INDEX and DROP INDEX.</p> <p>SQL_OU_PRIVILEGE_DEFINITION = Owners are supported in all privilege-definition statements: GRANT and REVOKE.</p>
SQL_POSITIONED_STATEMENTS	<p>A 32-bit bitmask that enumerates the supported positioned SQL statements.</p> <p>The following bitmasks are used to determine which statements are supported:</p> <p>SQL_PS_POSITIONED_DELETE  SQL_PS_POSITIONED_UPDATE  SQL_PS_SELECT_FOR_UPDATE</p>
SQL_PROCEDURE_TERM	A character string with the data-source vendor name for a procedure; for example, "database procedure," "stored procedure," or "procedure."
SQL_PROCEDURES	A character string: "Y" if the data source supports procedures and the driver supports the ODBC procedure invocation syntax; "N" otherwise.

(15 of 22)



InfoType	Returns
SQL_QUALIFIER_LOCATION	<p>A 16-bit integer value that indicates the position of the qualifier in a qualified table name:</p> <p>SQL_QL_START SQL_QL_END</p> <p>For example, a Text driver returns SQL_QL_START because the directory (qualifier) name is at the start of the table name, as in <b>/empdata/emp.dbf</b>.</p>
SQL_QUALIFIER_NAME_SEPARATOR	<p>A character string: the character or characters that the data source defines as the separator between a qualifier name and the qualified name element that follows it.</p>
SQL_QUALIFIER_TERM	<p>A character string with the data-source vendor name for a qualifier; for example, “database” or “directory.”</p>
SQL_QUALIFIER_USAGE	<p>A 32-bit bitmask that enumerates the statements in which qualifiers can be used.</p> <p>The following bitmasks are used to determine where qualifiers can be used:</p> <p>SQL_QU_DML_STATEMENTS = Qualifiers are supported in all Data Manipulation Language statements: SELECT, INSERT, UPDATE, DELETE, and, if supported, SELECT FOR UPDATE and positioned UPDATE and DELETE statements.</p> <p>SQL_QU_PROCEDURE_INVOCATION = Qualifiers are supported in the ODBC procedure invocation statement.</p> <p>SQL_QU_TABLE_DEFINITION = Qualifiers are supported in all table-definition statements: CREATE TABLE, CREATE VIEW, ALTER TABLE, DROP TABLE, and DROP VIEW.</p> <p>SQL_QU_INDEX_DEFINITION = Qualifiers are supported in all index-definition statements: CREATE INDEX and DROP INDEX.</p> <p>SQL_QU_PRIVILEGE_DEFINITION = Qualifiers are supported in all privilege-definition statements: GRANT and REVOKE.</p>

(16 of 22)

InfoType	Returns
SQL_QUOTED_IDENTIFIER_CASE	<p>A 16-bit integer value as follows:</p> <p>SQL_IC_UPPER = Quoted identifiers in SQL are case insensitive and are stored in uppercase in system catalog.</p> <p>SQL_IC_LOWER = Quoted identifiers in SQL are case insensitive and are stored in lowercase in system catalog.</p> <p>SQL_IC_SENSITIVE = Quoted identifiers in SQL are case sensitive and are stored in mixed case in system catalog.</p> <p>SQL_IC_MIXED = Quoted identifiers in SQL are not case sensitive and are stored in mixed case in system catalog.</p>
SQL_ROW_UPDATES	<p>A character string: "Y" if a keyset-driven or mixed cursor maintains row versions or values for all fetched rows and therefore can detect any changes made to a row by any user since the row was last fetched; otherwise, "N."</p>
SQL_SCROLL_CONCURRENCY	<p>A 32-bit bitmask that enumerates the concurrency control options supported for scrollable cursors.</p> <p>The following bitmasks are used to determine which options are supported:</p> <p>SQL_SCCO_READ_ONLY = Cursor is read only. No updates are allowed.</p> <p>SQL_SCCO_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated.</p> <p>SQL_SCCO_OPT_ROWVER = Cursor uses optimistic concurrency control, comparing row versions.</p> <p>SQL_SCCO_OPT_VALUES = Cursor uses optimistic concurrency control, comparing values.</p> <p>For information about cursor concurrency, see <a href="#">"Specifying Cursor Concurrency" on page 7-12.</a></p>

InfoType	Returns
SQL_SCROLL_OPTIONS	<p>A 32-bit bitmask that enumerates the scroll options supported for scrollable cursors.</p> <p>The following bitmasks are used to determine which options are supported:</p> <p>SQL_SO_FORWARD_ONLY = The cursor only scrolls forward.</p> <p>SQL_SO_STATIC = The data in the result set is static.</p> <p>SQL_SO_KEYSET_DRIVEN = The driver saves and uses the keys for every row in the result set.</p> <p>SQL_SO_DYNAMIC = The driver keeps the keys for every row in the rowset (the keyset size is the same as the rowset size).</p> <p>SQL_SO_MIXED = The driver 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.</p> <p>For information about scrollable cursors, see <a href="#">“Scrollable Cursors”</a> on page 7-10.</p>
SQL_SEARCH_PATTERN_ESCAPE	<p>A character string that specifies what the driver supports as an escape character that permits the use of the pattern-match metacharacters underscore ( <code>_</code> ) and percent ( <code>%</code> ) as valid characters in search patterns. This escape character applies only for those catalog function arguments that support search strings. If this string is empty, the driver does not support a search-pattern escape character.</p> <p>This <i>InfoType</i> is limited to catalog functions. For a description of the use of the escape character in search pattern strings, see <a href="#">“Search Pattern Arguments”</a> on page 13-8.</p>
SQL_SERVER_NAME	<p>A character string with the actual data-source-specific server name; useful when a data-source name is used during <b>SQLConnect</b>, <b>SQLDriverConnect</b>, and <b>SQLBrowseConnect</b>.</p>
SQL_SPECIAL_CHARACTERS	<p>A character string that contains all special characters (that is, all characters except a through z, A through Z, 0 through 9, and underscore) that can be used in an object name, such as a table, column, or index name, on the data source. For example, “#\$.^.”</p>

(18 of 22)

InfoType	Returns
SQL_STATIC_SENSITIVITY	<p>A 32-bit bitmask that enumerates whether changes made by an application to a static or keyset-driven cursor through positioned update or delete statements can be detected by that application:</p> <p>SQL_SS_ADDITIONS = Added rows are visible to the cursor; the cursor can scroll to these rows. Where these rows are added to the cursor is driver dependent.</p> <p>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.</p> <p>SQL_SS_UPDATES = Updates to rows are visible to the cursor; if the cursor scrolls from and returns to an updated row, the data returned by the cursor is the updated data, not the original data. Because updating key values in a keyset-driven cursor is considered to be deleting the existing row and adding a new row, this value is always returned for keyset-driven cursors.</p> <p>Whether an application can detect changes made to the result set by other users, including other cursors in the same application, depends on the cursor type. For more information, see <a href="#">“Scrollable Cursors” on page 7-10</a>.</p>
SQL_SYSTEM_FUNCTIONS	<p>A 32-bit bitmask that enumerates the scalar system functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which system functions are supported:</p> <p>SQL_FN_SYS_DBNAME  SQL_FN_SYS_IFNULL  SQL_FN_SYS_USERNAME</p>
SQL_TABLE_TERM	<p>A character string with the data-source vendor name for a table; for example, “table” or “file.”</p>

(19 of 22)

InfoType	Returns
SQL_TIMEDATE_ADD_INTERVALS	<p>A 32-bit bitmask that enumerates the time-stamp intervals supported by the driver and associated data source for the <code>TIMESTAMPADD</code> scalar function.</p> <p>The following bitmasks are used to determine which intervals are supported:</p> <ul style="list-style-type: none"> <li>SQL_FN_TSI_FRAC_SECOND</li> <li>SQL_FN_TSI_SECOND</li> <li>SQL_FN_TSI_MINUTE</li> <li>SQL_FN_TSI_HOUR</li> <li>SQL_FN_TSI_DAY</li> <li>SQL_FN_TSI_WEEK</li> <li>SQL_FN_TSI_MONTH</li> <li>SQL_FN_TSI_QUARTER</li> <li>SQL_FN_TSI_YEAR</li> </ul>
SQL_TIMEDATE_DIFF_INTERVALS	<p>A 32-bit bitmask that enumerates the time-stamp intervals supported by the driver and associated data source for the <code>TIMESTAMPDIFF</code> scalar function.</p> <p>The following bitmasks are used to determine which intervals are supported:</p> <ul style="list-style-type: none"> <li>SQL_FN_TSI_FRAC_SECOND</li> <li>SQL_FN_TSI_SECOND</li> <li>SQL_FN_TSI_MINUTE</li> <li>SQL_FN_TSI_HOUR</li> <li>SQL_FN_TSI_DAY</li> <li>SQL_FN_TSI_WEEK</li> <li>SQL_FN_TSI_MONTH</li> <li>SQL_FN_TSI_QUARTER</li> <li>SQL_FN_TSI_YEAR</li> </ul>

(20 of 22)

InfoType	Returns
SQL_TIMEDATE_FUNCTIONS	<p>A 32-bit bitmask that enumerates the scalar date and time functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which date and time functions are supported:</p> <ul style="list-style-type: none"> <li>SQL_FN_TD_CURDATE</li> <li>SQL_FN_TD_CURTIME</li> <li>SQL_FN_TD_DAYNAME</li> <li>SQL_FN_TD_DAYOFMONTH</li> <li>SQL_FN_TD_DAYOFWEEK</li> <li>SQL_FN_TD_DAYOFYEAR</li> <li>SQL_FN_TD_HOUR</li> <li>SQL_FN_TD_MINUTE</li> <li>SQL_FN_TD_MONTH )</li> <li>SQL_FN_TD_MONTHNAME )</li> <li>SQL_FN_TD_NOW</li> <li>SQL_FN_TD_QUARTER</li> <li>SQL_FN_TD_SECOND</li> <li>SQL_FN_TD_TIMESTAMPADD</li> <li>SQL_FN_TD_TIMESTAMPDIFF</li> <li>SQL_FN_TD_WEEK</li> <li>SQL_FN_TD_YEAR</li> </ul>
SQL_TXN_CAPABLE	<p>A 16-bit integer value that describes the transaction support in the driver or data source:</p> <p>SQL_TC_NONE = Transactions not supported.</p> <p>SQL_TC_DML = Transactions can contain only Data Manipulation Language (DML) statements (SELECT, INSERT, UPDATE, DELETE). Data Definition Language (DDL) statements encountered in a transaction cause an error.</p> <p>SQL_TC_DDL_COMMIT = Transactions can contain only DML statements. DDL statements (CREATE TABLE, DROP INDEX, and so on) encountered in a transaction cause the transaction to be committed.</p> <p>SQL_TC_DDL_IGNORE = Transactions can contain only DML statements. DDL statements encountered in a transaction are ignored.</p> <p>SQL_TC_ALL = Transactions can contain DDL statements and DML statements in any order.</p>

InfoType	Returns
SQL_TXN_ISOLATION_OPTION	<p>A 32-bit bitmask that enumerates the transaction-isolation levels available from the driver or data source. The following bitmasks are used in conjunction with the flag to determine which options are supported:</p> <p>SQL_TXN_READ_UNCOMMITTED  SQL_TXN_READ_COMMITTED  SQL_TXN_REPEATABLE_READ  SQL_TXN_SERIALIZABLE  SQL_TXN_VERSIONING</p> <p>For descriptions of these isolation levels, see <a href="#">“SQL_DEFAULT_TXN_ISOLATION”</a> on page 13-194.</p>
SQL_UNION	<p>A 32-bit bitmask that enumerates the support for the UNION clause:</p> <p>SQL_U_UNION = The data source supports the UNION clause.  SQL_U_UNION_ALL = The data source supports the ALL keyword in the UNION clause. (<b>SQLGetInfo</b> returns both SQL_U_UNION and SQL_U_UNION_ALL in this case.)</p>
SQL_USER_NAME	<p>A character string with the name used in a particular database, which can be different than login name.</p>

(22 of 22)

## Code Example

**SQLGetInfo** returns lists of supported options as a 32-bit bitmask in *rgbInfoValue*. The bitmask for each option is used with the flag to determine whether the option is supported.

For example, an application could use the following code to determine whether the SUBSTRING scalar function is supported by the driver associated with the *hdbc*:

```

UDWORD fFuncs;

SQLGetInfo(hdbc,
           SQL_STRING_FUNCTIONS,
           (PTR)&fFuncs,
           sizeof(fFuncs),
           NULL);

if (fFuncs & SQL_FN_STR_SUBSTRING) /* SUBSTRING supported */
    ...;
else                               /* SUBSTRING not supported */
    ...;

```

## Related Functions

For information about	See
Returning the setting of a connection option	<b>SQLGetConnectOption</b>
Determining if a driver supports a function	<b>SQLGetFunctions</b>
Returning the setting of a statement option	<b>SQLGetStmtOption</b>
Returning information about the data types of a data source	<b>SQLGetTypeInfo</b>



## SQLGetStmtOption

**SQLGetStmtOption** returns the current setting of a statement option.

### Syntax

```
RETCODE SQLGetStmtOption(hstmt, fOption, pvParam)
```

The **SQLGetStmtOption** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>fOption</i>	Input	Option to retrieve.
PTR	<i>pvParam</i>	Output	Value associated with <i>fOption</i> . Depending on the value of <i>fOption</i> , a 32-bit integer value or a pointer to a null-terminated character string will be returned in <i>pvParam</i> .

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetStmtOption** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetStmtOption** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
24000	Invalid cursor state	The argument <i>fOption</i> was <code>SQL_ROW_NUMBER</code> , but 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.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1011	Operation invalid at this time	The <i>fOption</i> argument was <code>SQL_GET_BOOKMARK</code> , and the value of the <code>SQL_USE_BOOKMARKS</code> statement option was <code>SQL_UB_OFF</code> .

(1 of 2)

SQLSTATE	Error	Description
S1092	Option type out of range	(DM) The value specified for the argument <i>fOption</i> was in the block of numbers reserved for ODBC connection and statement options but was not valid for the version of ODBC supported by the driver.
S1109	Invalid cursor position	The <i>fOption</i> argument was SQL_ROW_NUMBER, but the value in the <i>rgfRowStatus</i> array in <b>SQLExtendedFetch</b> for the current row was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The driver or data source does not support the value specified for the argument <i>fOption</i> .

(2 of 2)

## Usage

The following table lists statement options for which corresponding values can be returned but not set. For a list of options that can be set and retrieved, see “Usage” on page 13-288. If *fOption* specifies an option that returns a string, *pvParam* must be a pointer to storage for the string. The maximum length of the string is SQL\_MAX\_OPTION\_STRING\_LENGTH bytes, excluding the null-termination byte.

<i>fOption</i>	<i>pvParam</i> contents
SQL_GET_BOOKMARK UNSUPPORTED	INFORMIX-CLI does not support this option. The default is set to SQL_USE_BOOKMARKS=SQL_UB_OFF.
SQL_ROW_NUMBER	A 32-bit integer value that specifies 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, the driver returns 0.

## Related Functions

For information about	See
Returning the setting of a connection option	<b>SQLGetConnectOption</b>
Setting a connection option	<b>SQLSetConnectOption</b>
Setting a statement option	<b>SQLSetStmtOption</b>

## SQLGetTypeInfo

**SQLGetTypeInfo** returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set.

**Important:** *Applications must use the type names returned in the TYPE\_NAME column in ALTER TABLE and CREATE TABLE statements; they must not use the sample type names listed in Appendix C, “SQL Grammar.” SQLGetTypeInfo might return more than one row with the same value in the DATA\_TYPE column.*

## Syntax

```
RETCODE SQLGetTypeInfo(hstmt, fSqlType)
```

Level 1



The **SQLGetTypeInfo** function accepts the following arguments.:

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle for the result set.
SWORD	<i>fSqlType</i>	Input	<p>The SQL data type, which must be one of the following values:</p> <p>SQL_CHAR            SQL_DATE            SQL_DECIMAL            SQL_DOUBLE            SQL_INTEGER            SQL_LONGVARBINARY            SQL_LONGVARCHAR            SQL_REAL            SQL_SMALLINT            SQL_TIMESTAMP            SQL_VARCHAR</p> <p>SQL_ALL_TYPES specifies that information about all data types should be returned.</p> <p>For information on ODBC SQL data types, see <a href="#">Appendix C</a>. For information on how Informix data types map to ODBC SQL data types, see <a href="#">“Mapping Data Types” on page 1-15</a></p>

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLGetTypeInfo** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLGetTypeInfo** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1004	SQL data type out of range	(DM) The value specified for the argument <i>fSqlType</i> was in the block of numbers reserved for ODBC SQL data-type indicators but was not a valid ODBC SQL data-type indicator.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.

(1 of 2)

SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1C00	Driver not capable	The driver or data source does not support the value specified for the argument <i>fSqlType</i> .  The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

## Usage

**SQLGetTypeInfo** returns the results as a standard result set, ordered by DATA\_TYPE and TYPE\_NAME. The following table lists the columns in the result set.

The lengths of VARCHAR columns shown in the following table are maximums; the actual lengths depend on the data source.

Column Name	Data Type	Comments
TYPE_NAME	VARCHAR(128) not NULL	Data-source-dependent data-type name; for example, CHAR, VARCHAR, MONEY, LONG VARBINARY, or CHAR ( ) FOR BIT DATA. Applications must use this name in CREATE TABLE and ALTER TABLE statements.
DATA_TYPE	SMALLINT not NULL	SQL data type. For a list of valid ODBC SQL data types, see <a href="#">Appendix C</a> . For information on how Informix data types map to ODBC SQL data types, see <a href="#">“Mapping Data Types” on page 1-15</a> .
PRECISION	INTEGER	The maximum precision of the data type on the data source. NULL is returned for data types where precision is not applicable. For more information on precision, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a> .
LITERAL_PREFIX	VARCHAR(128)	Character or characters used to prefix a literal; for example, a single quote ( ' ) for character data types or 0x for binary data types; NULL is returned for data types where a literal prefix is not applicable.
LITERAL_SUFFIX	VARCHAR(128)	Character or characters used to terminate a literal; for example, a single quote ( ' ) for character data types; NULL is returned for data types where a literal suffix is not applicable.

(1 of 4)



Column Name	Data Type	Comments
CREATE_PARAMS	VARCHAR(128)	Parameters for a data-type definition. For example, CREATE_PARAMS for DECIMAL would be "precision, scale;" CREATE_PARAMS for VARCHAR would equal "max length;" NULL is returned if there are no parameters for the data type definition; for example, INTEGER. The driver supplies the CREATE_PARAMS text in the language of the country where it is used.
NULLABLE	SMALLINT not NULL	Whether the data type accepts a NULL value:  SQL_NO_NULLS if the data type does not accept NULL values.  SQL_NULLABLE if the data type accepts NULL values.  SQL_NULLABLE_UNKNOWN if it is not known if the column accepts NULL values.
CASE_SENSITIVE	SMALLINT not NULL	Whether a character data type is case sensitive in collations and comparisons:  TRUE if the data type is a character data type and is case sensitive.  FALSE if the data type is not a character data type or is not case sensitive.

(2 of 4)

Column Name	Data Type	Comments
SEARCHABLE	SMALLINT not NULL	<p>How the data type is used in a WHERE clause:</p> <p>SQL_UNSEARCHABLE if the data type cannot be used in a WHERE clause.</p> <p>SQL_LIKE_ONLY if the data type can be used in a WHERE clause only with the LIKE predicate.</p> <p>SQL_ALL_EXCEPT_LIKE if the data type can be used in a WHERE clause with all comparison operators except LIKE.</p> <p>SQL_SEARCHABLE if the data type can be used in a WHERE clause with any comparison operator.</p>
UNSIGNED_ATTRIBUTE	SMALLINT	<p>Whether the data type is unsigned:</p> <p>TRUE if the data type is unsigned.</p> <p>FALSE if the data type is signed.</p> <p>NULL is returned if the attribute is not applicable to the data type or the data type is not numeric.</p>
MONEY	SMALLINT not NULL	<p>Whether the data type is a money data type:</p> <p>TRUE if it is a money data type.</p> <p>FALSE if it is not.</p>
AUTO_INCREMENT	SMALLINT	<p>Whether the data type is autoincrementing:</p> <p>TRUE if the data type is autoincrementing.</p> <p>FALSE if the data type is not autoincrementing.</p> <p>NULL is returned if the attribute is not applicable to the data type or the data type is not numeric.</p> <p>An application can insert values into a column having this attribute, but cannot update the values in the column.</p>

Column Name	Data Type	Comments
LOCAL_TYPE_NAME	VARCHAR(128)	Localized version of the data-source-dependent name of the data type. NULL is returned if a localized name is not supported by the data source. This name is intended for display only, as in dialog boxes.
MINIMUM_SCALE	SMALLINT	The minimum scale of the data type on the data source. If a data type has a fixed scale, the MINIMUM_SCALE and MAXIMUM_SCALE columns both contain this value. For example, an SQL_TIMESTAMP column might have a fixed scale for fractional seconds. NULL is returned where scale is not applicable. For more information, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a> .
MAXIMUM_SCALE	SMALLINT	The maximum scale of the data type on the data source. NULL is returned where scale is not applicable. If the maximum scale is not defined separately on the data source but is instead defined to be the same as the maximum precision, this column contains the same value as the PRECISION column. For more information, see <a href="#">“Precision, Scale, Length, and Display Size” on page C-8</a> .

(4 of 4)

Attribute information can apply to data types or to specific columns in a result set. **SQLGetTypeInfo** returns information about attributes associated with data types; **SQLColAttributes** returns information about attributes associated with columns in a result set.

## Related Functions

---

<b>For information about</b>	<b>See</b>
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning information about a column in a result set	<b>SQLColAttributes</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning information about a driver or data source	<b>SQLGetInfo</b>

---

## Level 2

## SQLMoreResults

**SQLMoreResults** determines whether more results are available on an *hstmt* that contains SELECT, UPDATE, INSERT, or DELETE statements and, if so, initializes processing for those results.

### Syntax

```
RETCODE SQLMoreResults(hstmt)
```

The **SQLMoreResults** function accepts the following argument:

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_NO\_DATA\_FOUND, SQL\_ERROR, or SQL\_INVALID\_HANDLE.

## Diagnostics

When **SQLMoreResults** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLMoreResults** and explains each one in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , <code>SQL_QUERY_TIMEOUT</code> .

## Usage

SELECT statements return result sets. UPDATE, INSERT, and DELETE statements return a count of affected rows. If any of these statements are batched, submitted with arrays of parameters, or in procedures, they can return multiple result sets or counts.

If another result set or count is available, **SQLMoreResults** returns `SQL_SUCCESS` and initializes the result set or count for additional processing. After calling **SQLMoreResults** for SELECT statements, an application can call functions to determine the characteristics of the result set and to retrieve data from the result set. After calling **SQLMoreResults** for UPDATE, INSERT, or DELETE statements, an application can call **SQLRowCount**.

If all results have been processed, **SQLMoreResults** returns `SQL_NO_DATA_FOUND`.

If there is a current result set with un fetched rows, **SQLMoreResults** discards that result set and makes the next result set or count available.

If a batch of statements or a procedure mixes other SQL statements with SELECT, UPDATE, INSERT, and DELETE statements, these other statements do not affect **SQLMoreResults**.

For additional information about the valid sequencing of result-processing functions, see Appendix B, “ODBC State Transition Tables.”

## Related Functions

For information about	See
Canceling statement processing	<b>SQLCancel</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Fetching part or all of a column of data	<b>SQLGetData</b>

## SQLNativeSql

**SQLNativeSql** returns the SQL string translated by the driver.

### Syntax

```
RETCODE SQLNativeSql(hdbc, szSqlStrIn, cbSqlStrIn, szSqlStr,
                    cbSqlStrMax, pcbSqlStr)
```

The **SQLNativeSql** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
UCHAR FAR *	<i>szSqlStrIn</i>	Input	SQL text string to be translated.
SDWORD	<i>cbSqlStrIn</i>	Input	Length of <i>szSqlStrIn</i> text string.
UCHAR FAR *	<i>szSqlStr</i>	Output	Pointer to storage for the translated SQL string.
SDWORD	<i>cbSqlStrMax</i>	Input	Maximum length of the <i>szSqlStr</i> buffer.
SDWORD FAR*	<i>pcbSqlStr</i>	Output	The total number of bytes (excluding the null-termination byte) available to return in <i>szSqlStr</i> . If the number of bytes available to return is greater than or equal to <i>cbSqlStrMax</i> , the translated SQL string in <i>szSqlStr</i> is truncated to <i>cbSqlStrMax</i> - 1 bytes.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE



## Diagnostics

When **SQLNativeSql** returns either `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLNativeSql** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	Data truncated	The buffer <i>szSqlStr</i> was not large enough to return the entire SQL string, so the SQL string was truncated. The argument <i>pcbSqlStr</i> contains the length of the untruncated SQL string (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
08003	Connection not open	The <i>hdbc</i> was not in a connected state.
37000	Syntax error or access violation	The argument <i>szSqlStrIn</i> contained an SQL statement that was not preparable or contained a syntax error.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.

(1 of 2)

SQLSTATE	Error	Description
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	(DM) The argument <i>szSqlStrIn</i> was a null pointer.
S1090	Invalid string or buffer length	(DM) The argument <i>cbSqlStrIn</i> was less than 0, but not equal to SQL_NTS.  (DM) The argument <i>cbSqlStrMax</i> was less than 0, and the argument <i>szSqlStr</i> was not a null pointer.

(2 of 2)

## Usage

The following example shows what **SQLNativeSql** might return for an input SQL string that contains the scalar function LENGTH:

```
SELECT {fn LENGTH(NAME)} FROM EMPLOYEE
```

An INFORMIX-CLI driver might return the following translated SQL string:

```
SELECT length(NAME) FROM EMPLOYEE
```

## Related Functions

None.

## SQLNumParams

**SQLNumParams** returns the number of parameters in an SQL statement.

### Syntax

```
RETCODE SQLNumParams(hstmt, pcpar)
```

The **SQLNumParams** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
SWORD FAR *	<i>pcpar</i>	Output	Number of parameters in the statement

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

### Diagnostics

When **SQLNumParams** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLNumParams** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) The function was called prior to calling <b>SQLPrepare</b> or <b>SQLExecDirect</b> for the <i>hstmt</i> . (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , <b>SQL_QUERY_TIMEOUT</b> .

## Usage

**SQLNumParams** can be called only after **SQLPrepare** has been called.

If the statement associated with *hstmt* does not contain parameters, **SQLNumParams** sets *pcpar* to 0.

## Related Functions

For information about	See
Assigning storage for a parameter	<b>SQLBindParameter</b>

## SQLNumResultCols

**SQLNumResultCols** returns the number of columns in a result set.

### Syntax

```
RETCODE SQLNumResultCols(hstmt, pccol)
```

The **SQLNumResultCols** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
SQLINTEGER *	<i>pccol</i>	Output	Number of columns in the result set

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

Core

## Diagnostics

When **SQLNumResultCols** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLNumResultCols** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) The function was called prior to calling <b>SQLPrepare</b> or <b>SQLExecDirect</b> for the <i>hstmt</i> . (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , <code>SQL_QUERY_TIMEOUT</code> .

**SQLNumResultCols** can return any SQLSTATE that can be returned by **SQLPrepare** or **SQLExecute** when **SQLNumResultCols** is called after **SQLPrepare** and before **SQLExecute** is called, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

## Usage

**SQLNumResultCols** can be called successfully only when the *hstmt* is in the prepared, executed, or positioned state.

If the statement associated with *hstmt* does not return columns, **SQLNumResultCols** sets *pccol* to 0.

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning information about a column in a result set	<b>SQLColAttributes</b>
Returning information about a column in a result set	<b>SQLDescribeCol</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Fetching part or all of a column of data	<b>SQLGetData</b>
Setting cursor scrolling options	<b>SQLSetScrollOptions</b>

## SQLParamData

**SQLParamData** is used with **SQLPutData** to supply parameter data when a statement executes.

### Syntax

```
RETCODE SQLParamData(hstmt, prgbValue)
```

The **SQLParamData** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
PTR FAR *	<i>prgbValue</i>	Output	Pointer to storage for the value specified for the <i>rgbValue</i> argument in <b>SQLBindParameter</b> (for parameter data) or the address of the <i>rgbValue</i> buffer specified in <b>SQLBindCol</b> (for column data).

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_NEED\_DATA,  
SQL\_STILL\_EXECUTING, SQL\_ERROR, or  
SQL\_INVALID\_HANDLEINFORMIX-CLI



## Diagnostics

When **SQLParamData** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLParamData** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
22026	String data, length mismatch	The <code>SQL_NEED_LONG_DATA_LEN</code> information type in <b>SQLGetInfo</b> was “Y” and less data was sent for a long parameter (the data type was <code>SQL_LONGVARCHAR</code> , <code>SQL_LONGVARBINARY</code> , or a long, data-source-specific data type) than was specified with the <i>pcbValue</i> argument in <b>SQLBindParameter</b> .
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

(1 of 2)

SQLSTATE	Error	Description
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.  <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. <b>SQLCancel</b> was called before data was sent for all data-at-execution parameters or columns.
S1010	Function-sequence error	(DM) The previous function call was not a call to <b>SQLExecDirect</b> or <b>SQLExecute</b> where the return code was SQL_NEED_DATA or a call to <b>SQLPutData</b> .  The previous function call was a call to <b>SQLParamData</b> .
S1T00	Time-out expired	The time-out period expired before the data source completed processing the parameter value. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

If **SQLParamData** is called while sending data for a parameter in an SQL statement, it can return any SQLSTATE that can be returned by the function that was called to execute the statement (**SQLExecute** or **SQLExecDirect**).

## Usage

For an explanation of how data-at-execution parameter data is passed at statement-execution time, see [“Passing Parameter Values” on page 13-34](#).

## Code Example

See **SQLPutData**.

## Related Functions

For information about	See
Canceling statement processing	<b>SQLCancel</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Sending parameter data at execution time	<b>SQLPutData</b>
Assigning storage for a parameter	<b>SQLBindParameter</b>

## SQLParamOptions

**SQLParamOptions** allows an application to specify multiple values for the set of parameters assigned by **SQLBindParameter**. The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. For example, an application can specify three sets of values for the set of parameters associated with an INSERT statement, and then execute the INSERT statement once to perform the three insert operations.

### Syntax

```
RETCODE SQLParamOptions(hstmt, crow, pirow)
```

Level 2

The **SQLParamOptions** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UDWORD	<i>crow</i>	Input	Number of values for each parameter. If <i>crow</i> is greater than 1, the <i>rgbValue</i> argument in <b>SQLBindParameter</b> points to an array of parameter values, and <i>pcbValue</i> points to an array of lengths.
UDWORD FAR *	<i>pirow</i>	Input	Pointer to storage for the current row number. As each row of parameter values is processed, <i>pirow</i> is set to the number of that row. No row number will be returned if <i>pirow</i> is set to a null pointer.

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE.

## Diagnostics

When **SQLParamOptions** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value may be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLParamOptions** and explains each one in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1107	Row value out of range	(DM) The value specified for the argument <i>crow</i> was equal to 0.

## Usage

As a statement executes, the driver sets *pirow* to the number of the current row of parameter values; the first row is row number 1. The contents of *pirow* can be used as follows:

- When **SQLParamData** returns `SQL_NEED_DATA` for data-at-execution parameters, the application can access the value in *pirow* to determine which row of parameters is being executed.
- When **SQLExecute** or **SQLExecDirect** returns an error, the application can access the value in *pirow* to find out which row of parameters failed.
- When **SQLExecute**, **SQLExecDirect**, **SQLParamData**, or **SQLPutData** succeed, the value in *pirow* is set to *crow* (the total number of rows of parameters processed).

## Code Example

In the following example, an application specifies an array of parameter values with `SQLBindParameter` and `SQLParamOptions`. It then inserts those values into a table with a single `INSERT` statement and checks for any errors. If the first row fails, the application rolls back all changes. If any other row fails, the application commits the transaction, skips the failed row, rebinds the remaining parameters, and continues processing. (Note that `irow` is 1-based and `szData[]` is 0-based, so the `irow` entry of `szData[]` is skipped by rebinding at `szData[irow]`.)

```
#define CITY_LEN 256
SDWORD cbValue[] = {SQL_NTS, SQL_NTS, SQL_NTS, SQL_NTS, SQL_NTS};
UCHAR  szData[CITY_LEN] = {"Boston","New York","Keokuk","Seattle",
                           "Eugene"};

UDWORD irow;
SQLSetConnectOption(hdbc, SQL_AUTOCOMMIT, 0);
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_DEFAULT, SQL_CHAR,
                 CITY_LEN, 0, szData, 0, cbValue);
SQLPrepare(hstmt, "INSERT INTO CITIES VALUES (?)", SQL_NTS);
SQLParamOptions(hstmt, 5, &irow);

while (TRUE) {

    retcode = SQLExecute(hstmt);

    /* Done if execution was successful */

    if (retcode != SQL_ERROR) {
        break;
    }
    /* On an error, print the error. If the error is in row 1, roll */
    /* back the transaction and quit. If the error is in another */
    /* row, commit the transaction and, unless the error is in the */
    /* last row, rebind to the next row and continue processing. */

    show_error();
    if (irow == 1) {
        SQLTransact(henv, hstmt, SQL_ROLLBACK);
        break;
    } else {
        SQLTransact(henv, hstmt, SQL_COMMIT);
        if (irow == 5) {
            break;
        } else {
            SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT,
                             SQL_C_DEFAULT, SQL_CHAR, CITY_LEN, 0,
                             szData[irow], 0, cbValue[irow]);
            SQLParamOptions(hstmt, 5-irow, &irow);
        }
    }
}
}
```

## Related Functions

For information about	See
Assigning storage for a parameter	<b>SQLBindParameter</b>

## SQLPrepare

SQLPrepare prepares an SQL string for execution.

### Syntax

```
RETCODE SQLPrepare(hstmt, szSqlStr, cbSqlStr)
```

The **SQLPrepare** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
UCHAR FAR *	<i>szSqlStr</i>	Input	SQL text string
SDWORD	<i>cbSqlStr</i>	Input	Length of <i>szSqlStr</i>

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE



## Diagnostics

When **SQLPrepare** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLPrepare** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
21S01	Insert value list does not match column list.	The argument <i>szSqlStr</i> contained an INSERT statement, and the number of values to be inserted did not match the degree of the derived table.
21S02	Degree of derived table does not match column list.	The argument <i>szSqlStr</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.
22005	Error in assignment	The argument <i>szSqlStr</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.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
34000	Invalid cursor name	The argument <i>szSqlStr</i> contained a positioned DELETE or a positioned UPDATE, and the cursor referenced by the statement being prepared was not open.
37000	Syntax error or access violation	The argument <i>szSqlStr</i> contained an SQL statement that was not preparable or contained a syntax error.

(1 of 4)

SQLSTATE	Error	Description
42000	Syntax error or access violation	The argument <i>szSqlStr</i> contained a statement for which the user did not have the required privileges.
S0001	Base table or view already exists	The argument <i>szSqlStr</i> contained a CREATE TABLE or CREATE VIEW statement, but the table name or view name specified already exists.
S0002	Base table not found	<p>The argument <i>szSqlStr</i> contained a DROP TABLE or a DROP VIEW statement, but the specified table name or view name did not exist.</p> <p>The argument <i>szSqlStr</i> contained an ALTER TABLE statement, but the specified table name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE VIEW statement, but a table name or view name defined by the query specification did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE INDEX statement, but the specified table name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a GRANT or REVOKE statement, but the specified table name or view name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a SELECT statement, but a specified table name or view name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a DELETE, INSERT, or UPDATE statement, but the specified table name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE TABLE statement, and a table specified in a constraint (referencing a table other than the one being created) did not exist.</p>

(2 of 4)

SQLSTATE	Error	Description
S0011	Index already exists.	The argument <i>szSqlStr</i> contained a CREATE INDEX statement, but the specified index name already existed.
S0012	Index not found	The argument <i>szSqlStr</i> contained a DROP INDEX statement, but the specified index name did not exist.
S0021	Column already exists.	The argument <i>szSqlStr</i> contained an ALTER TABLE statement, and the column specified in the ADD clause is not unique or identifies an existing column in the base table.
S0022	Column not found	<p>The argument <i>szSqlStr</i> contained a CREATE INDEX statement, and one or more of the column names specified in the column list did not exist.</p> <p>The argument <i>szSqlStr</i> contained a GRANT or REVOKE statement, and a specified column name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a SELECT, DELETE, INSERT, or UPDATE statement, and a specified column name did not exist.</p> <p>The argument <i>szSqlStr</i> contained a CREATE TABLE statement, and a column specified in a constraint (referencing a table other than the one being created) did not exist.</p>
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

(3 of 4)

SQLSTATE	Error	Description
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1009	Invalid argument value	(DM) The argument <i>szSqlStr</i> was a null pointer.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The argument <i>cbSqlStr</i> was less than or equal to 0, but not equal to SQL_NTS.
S1C00	Driver not capable	The cursor/concurrency combination is invalid.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(4 of 4)

## Usage

The application calls **SQLPrepare** to send an SQL statement to the data source for preparation. The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL string at the appropriate position.

**Tip:** If an application uses **SQLPrepare** to prepare and **SQLExecute** to submit a **COMMIT** or **ROLLBACK** statement, it is not interoperable among DBMS products. To commit or roll back a transaction, call **SQLTransact**.



The driver modifies the statement to use the form of SQL used by the data source and then submits it to the data source for preparation. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL. (For a description of SQL statement grammar, see [Appendix B](#).) For the driver, an *hstmt* is similar to a statement identifier in embedded SQL code. If the data source supports statement identifiers, the driver can send a statement identifier and parameter values to the data source.

Once a statement is prepared, the application uses *hstmt* to refer to the statement in later function calls. The prepared statement associated with the *hstmt* might be reexecuted by calling **SQLExecute** until the application frees the *hstmt* with a call to **SQLFreeStmt** with the `SQL_DROP` option or until the *hstmt* is used in a call to **SQLPrepare**, **SQLExecDirect**, or one of the catalog functions (**SQLColumns**, **SQLTables**, and so on). Once the application prepares a statement, it can request information about the format of the result set.

Some drivers cannot return syntax errors or access violations when the application calls **SQLPrepare**. A driver might handle syntax errors and access violations, only syntax errors, or neither syntax errors nor access violations. Therefore, an application must be able to handle these conditions when it calls subsequent related functions such as **SQLNumResultCols**, **SQLDescribeCol**, **SQLColAttributes**, and **SQLExecute**.

Depending on the capabilities of the driver and data source and on whether the application has called **SQLBindParameter**, parameter information (such as data types) might be checked when the statement is prepared or when it is executed. For maximum interoperability, an application should unbind all parameters that applied to an old SQL statement before it prepares a new SQL statement on the same *hstmt*. This action prevents errors that are caused by old parameter information being applied to the new statement.



**Warning:** *Committing or rolling back a transaction, either by calling **SQLTransact** or by using the `SQL_AUTOCOMMIT` connection option, can cause the data source to delete the access plans for all *hstmts* on an *hdbc*. For more information, see “[SQL\\_CURSOR\\_COMMIT\\_BEHAVIOR](#)” and “[SQL\\_CURSOR\\_ROLLBACK\\_BEHAVIOR](#)” on page 13-192.*

## Code Example

See **SQLBindParameter**, **SQLParamOptions**, and **SQLPutData**.

## Related Functions

<b>For information about</b>	<b>See</b>
Allocating a statement handle	<b>SQLAllocStmt</b>
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Returning the number of rows affected by a statement	<b>SQLRowCount</b>
Setting a cursor name	<b>SQLSetCursorName</b>
Assigning storage for a parameter	<b>SQLBindParameter</b>
Executing a commit or rollback operation	<b>SQLTransact</b>

## SQLPrimaryKeys

**SQLPrimaryKeys** returns the column names that comprise the primary key for a table. The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.

### Syntax

```
RETCODE SQLPrimaryKeys(hstmt, szTableQualifier,
    cbTableQualifier, szTableOwner, cbTableOwner, szTableName,
    cbTableName)
```

The **SQLPrimaryKeys** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szTableQualifier</i>	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i> .
UCHAR FAR *	<i>szTableOwner</i>	Input	Table owner. If a driver supports owners for some tables but not for others, as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i> .
UCHAR FAR *	<i>szTableName</i>	Input	Table name.
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLPrimaryKeys** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLPrimaryKeys** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.

(1 of 2)



SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name-length arguments was less than 0, but not equal to <code>SQL_NTS</code> .  The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name.
S1C00	Driver not capable	A table qualifier was specified, but the driver or data source does not support qualifiers.  A table owner was specified, but the driver or data source does not support owners.  The combination of the current settings of the <code>SQL_CONCURRENCY</code> and <code>SQL_CURSOR_TYPE</code> statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the requested result set. The time-out period is set through <b>SQLSetStmtOption</b> , <code>SQL_QUERY_TIMEOUT</code> .

(2 of 2)

## Usage

**SQLPrimaryKeys** returns the results as a standard result set, ordered by `TABLE_QUALIFIER`, `TABLE_OWNER`, `TABLE_NAME`, and `KEY_SEQ`. The following table lists the columns in the result set.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the **TABLE\_QUALIFIER**, **TABLE\_OWNER**, **TABLE\_NAME**, and **COLUMN\_NAME** columns, call **SQLGetInfo** with the **SQL\_MAX\_QUALIFIER\_NAME\_LEN**, **SQL\_MAX\_OWNER\_NAME\_LEN**, **SQL\_MAX\_TABLE\_NAME\_LEN**, and **SQL\_MAX\_COLUMN\_NAME\_LEN** options.

Column Name	Data Type	Comments
<b>TABLE_QUALIFIER</b>	VARCHAR(128)	Primary-key table-qualifier identifier; NULL if not applicable to the data source.
<b>TABLE_OWNER</b>	VARCHAR(128)	Primary-key table-owner identifier; NULL if not applicable to the data source.
<b>TABLE_NAME</b>	VARCHAR(128) not NULL	Primary-key table identifier.
<b>COLUMN_NAME</b>	VARCHAR(128) not NULL	Primary-key column identifier.
<b>KEY_SEQ</b>	SMALLINT not NULL	Column sequence number in key (starting with 1).
<b>PK_NAME</b>	VARCHAR(128)	Primary-key identifier. NULL if not applicable to the data source.

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning table statistics and indexes	<b>SQLStatistics</b>

## Level 2

## SQLProcedures

**SQLProcedures** returns the list of procedure names stored in a specific data source. *Procedure* is a generic term used to describe an *executable object*, or a named entity that can be invoked using input and output parameters, and which can return result sets similar to the results returned by SQL SELECT expressions.

### Syntax

```
RETCODE SQLProcedures(hstmt, szProcQualifier,
  cbProcQualifier, szProcOwner, cbProcOwner, szProcName,
  cbProcName)
```

The **SQLProcedures** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szProcQualifier</i>	Input	Procedure qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbProcQualifier</i>	Input	Length of <i>szProcQualifier</i> .
UCHAR FAR *	<i>szProcOwner</i>	Input	String search pattern for procedure-owner names. If a driver supports owners for some procedures but not for others, as when the driver retrieves data from different DBMSs, an empty string ("") denotes those procedures that do not have owners.

(1 of 2)

Type	Argument	Use	Description
SWORD	<i>cbProcOwner</i>	Input	Length of <i>szProcOwner</i> .
UCHAR FAR *	<i>szProcName</i>	Input	String search pattern for procedure names.
SWORD	<i>cbProcName</i>	Input	Length of <i>szProcName</i> .

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE.

## Diagnostics

When **SQLProcedures** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLProcedures** and explains each one in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.

(1 of 3)

SQLSTATE	Error	Description
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name-length arguments was less than 0 but not equal to SQL_NTS.  The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name.

(2 of 3)

SQLSTATE	Error	Description
S1C00	Driver not capable	<p>A procedure qualifier was specified, but the driver or data source does not support qualifiers.</p> <p>A procedure owner was specified, but the driver or data source does not support owners. A string search pattern was specified for the procedure owner or procedure name, but the data source does not support search patterns for one or more of those arguments.</p> <p>The combination of the current settings of the <code>SQL_CONCURRENCY</code> and <code>SQL_CURSOR_TYPE</code> statement options was not supported by the driver or data source.</p>
S1T00	Time-out expired	<p>The time-out period expired before the data source returned the requested result set. The time-out period is set through <code>SQLSetStmtOption</code>, <code>SQL_QUERY_TIMEOUT</code>.</p>

(3 of 3)

## Usage

`SQLProcedures` lists all procedures in the requested range. A user might or might not have permission to execute any of these procedures. To check accessibility, an application can call `SQLGetInfo` and check the `SQL_ACCESSIBLE_PROCEDURES` information value. Otherwise, the application must be able to handle a situation in which the user selects a procedure that it cannot execute.



**Important:** *`SQLProcedures` might not return all procedures. Applications can use any valid procedure, regardless of whether it is returned by `SQLProcedures`.*

`SQLProcedures` returns the results as a standard result set, ordered by `PROCEDURE_QUALIFIER`, `PROCEDURE_OWNER`, and `PROCEDURE_NAME`. The following table lists the columns in the result set.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the PROCEDURE\_QUALIFIER, PROCEDURE\_OWNER, and PROCEDURE\_NAME columns, an application can call **SQLGetInfo** with the SQL\_MAX\_QUALIFIER\_NAME\_LEN, SQL\_MAX\_OWNER\_NAME\_LEN, and SQL\_MAX\_PROCEDURE\_NAME\_LEN options.

Column Name	Data Type	Comments
PROCEDURE_QUALIFIER	VARCHAR(128)	Procedure qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some procedures but not for others, as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those procedures that do not have qualifiers.
PROCEDURE_OWNER	VARCHAR(128)	Procedure owner identifier; NULL if not applicable to the data source. If a driver supports owners for some procedures but not for others, as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those procedures that do not have owners.
PROCEDURE_NAME	VARCHAR(128) not NULL	Procedure identifier.
NUM_INPUT_PARAMS	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
NUM_OUTPUT_PARAMS	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
NUM_RESULT_SETS	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.

(1 of 2)

Column Name	Data Type	Comments
REMARKS	VARCHAR(254)	A description of the procedure.
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 is, it does not have a return value. SQL_PT_FUNCTION: The returned object is a function; that is, it has a return value.

(2 of 2)



The *szProcOwner* and *szProcName* arguments accept search patterns. For more information about valid search patterns, see [“Search Pattern Arguments” on page 13-8](#).



## Code Example

In this example, an application uses the procedure **AddEmployee** to insert data into the **EMPLOYEE** table. The procedure contains input parameters for **NAME**, **AGE**, and **BIRTHDAY** columns. It also contains one output parameter that returns a remark about the new employee. The example also shows the use of a return value from a stored procedure. For the return value and each parameter in the procedure, the application calls **SQLBindParameter** to specify the ODBC C data type and the SQL data type of the parameter and to specify the storage location and length of the parameter. The application assigns data values to the storage locations for each parameter and calls **SQLExecDirect** to execute the procedure. If **SQLExecDirect** returns **SQL\_SUCCESS** or **SQL\_SUCCESS\_WITH\_INFO**, the return value and the value of each output or input/output parameter is automatically put into the storage location defined for the parameter in **SQLBindParameter**.

```
#define NAME_LEN 30
#define REM_LEN 128

UCHAR      szName[NAME_LEN], szRemark[REM_LEN];
SWORD      sAge, sEmpId;
SDWORD     cbEmpId, cbName, cbAge = 0, cbBirthday = 0, cbRemark;
DATE_STRUCT dsBirthday;
/* Define parameter for return value (Employee ID) from procedure. */

SQLBindParameter(hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER,
                 0, 0, &sEmpId, 0, &cbEmpId);

/* Define data types and storage locations for Name, Age, Birthday */
/* input parameter data. */

SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
                 NAME_LEN, 0, szName, 0, &cbName);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_SSHORT, SQL_SMALLINT,
                 0, 0, &sAge, 0, &cbAge);
SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_DATE, SQL_DATE,
                 0, 0, &dsBirthday, 0, &cbBirthday);

/* Define data types and storage location for Remark output parameter */

SQLBindParameter(hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR,
                 REM_LEN, 0, szRemark, REM_LEN, &cbRemark);

strcpy(szName, "Smith, John D."); /* Specify first row of */
sAge = 40; /* parameter data. */
dsBirthday.year = 1952;
dsBirthday.month = 2;
dsBirthday.day = 29;
cbName = SQL_NTS;

/* Execute procedure with first row of data. After the procedure */
```

```

/* is executed, sEmpId and szRemark will have the values      */
/* returned by AddEmployee.                                   */

retcode = SQLExecDirect(hstmt, "{?=call AddEmployee(?,?,?,?)",SQL_NTS);

strcpy(szName, "Jones, Bob K.");      /* Specify second row of */
sAge = 52;                             /* parameter data        */
dsBirthday.year = 1940;
dsBirthday.month = 3;
dsBirthday.day = 31;

/* Execute procedure with second row of data. After the procedure */
/* is executed, sEmpId and szRemark will have the new values      */
/* returned by AddEmployee.                                       */

retcode = SQLExecDirect(hstmt,
                        "{?=call AddEmployee(?,?,?,?)", SQL_NTS);

```

## Related Functions

For information about	See
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning information about a driver or data source	<b>SQLGetInfo</b>
Syntax for invoking stored procedures	<b>Chapter 6, “Executing SQL Statements”</b>

## Level 1

## SQLPutData

**SQLPutData** allows an application to send data for a parameter or column to the driver at statement execution time. This function can send character or binary data values in parts to a column with a character, binary, or data-source-specific data type (for example, parameters of `SQL_LONGVARBINARY` or `SQL_LONGVARCHAR`).

### Syntax

```
RETCODE SQLPutData(hstmt, rgbValue, cbValue)
```

The **SQLPutData** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
PTR	<i>rgbValue</i>	Input	Pointer to storage for the actual data for the parameter or column. The data must use the C data type specified in the <i>fCType</i> argument of <b>SQLBindParameter</b> (for parameter data) or <b>SQLBindCol</b> (for column data).
SDWORD	<i>cbValue</i>	Input	Length of <i>rgbValue</i> . Specifies the amount of data sent in a call to <b>SQLPutData</b> . The amount of data can vary with each call for a given parameter or column. <i>cbValue</i> is ignored unless it is <code>SQL_NTS</code> , <code>SQL_NULL_DATA</code> , or <code>SQL_DEFAULT_PARAM</code> ; the C data type specified in <b>SQLBindParameter</b> or <b>SQLBindCol</b> is <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> ; or the C data type is <code>SQL_C_DEFAULT</code> and the default C data type for the specified SQL data type is <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> . For all other types of C data, if <i>cbValue</i> is not <code>SQL_NULL_DATA</code> or <code>SQL_DEFAULT_PARAM</code> , the driver assumes that the size of <i>rgbValue</i> is the size of the C data type specified with <i>fCType</i> and sends the entire data value. For more information, see <a href="#">“Converting Data from C to SQL Data Types”</a> on page C-24.

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLPutData** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLPutData** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The data sent for a character or binary parameter or column in one or more calls to <b>SQLPutData</b> exceeded the maximum length of the associated character or binary column.  The fractional part of the data sent for a numeric or bit parameter or column was truncated.  Time-stamp data sent for a date or time parameter or column was truncated.
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
22001	String data right truncation	The SQL_NEED_LONG_DATA_LEN information type in <b>SQLGetInfo</b> was “Y” and more data was sent for a long parameter (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data-source-specific data type) than was specified with the <i>pcbValue</i> argument in <b>SQLBindParameter</b> .

(1 of 3)

SQLSTATE	Error	Description
22003	Numeric value out of range	<p><b>SQLPutData</b> was called more than once for a parameter or column, and it was not being used to send character C data to a column with a character, binary, or data-source-specific data type or to send binary C data to a column with a character, binary, or data-source-specific data type.</p> <p>The data sent for a numeric parameter or column caused the whole (as opposed to fractional) part of the number to be truncated when assigned to the associated table column.</p>
22005	Error in assignment	The data sent for a parameter or column was incompatible with the data type of the associated table column.
22008	Datetime-field overflow	The data sent for a date, time, or time-stamp parameter or column was respectively, an invalid date, time, or time stamp.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver could not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	<p>The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.</p> <p><b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. <b>SQLCancel</b> was called before data was sent for all data-at-execution parameters or columns.</p>

(2 of 3)

SQLSTATE	Error	Description
S1009	Invalid argument value	(DM) The argument <i>rgbValue</i> was a null pointer, and the argument <i>cbValue</i> was not 0, SQL_DEFAULT_PARAM, or SQL_NULL_DATA.
S1010	Function-sequence error	(DM) The previous function call was not a call to <b>SQLPutData</b> or <b>SQLParamData</b> .  The previous function call was a call to <b>SQLExecDirect</b> or <b>SQLExecute</b> where the return code was SQL_NEED_DATA.
S1090	Invalid string or buffer length	The argument <i>rgbValue</i> was not a null pointer, and the argument <i>cbValue</i> was less than 0 but not equal to SQL_NTS or SQL_NULL_DATA.
S1T00	Time-out expired	The time-out period expired before the data source completed processing the parameter value. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(3 of 3)

## Usage

For an explanation of how data-at-execution parameter data passes when a statement executes, see [“Passing Parameter Values” on page 13-34](#). For an explanation of how data-at-execution column data is updated or added, see [“SQLSetScrollOptions” on page 13-285](#).



**Important:** An application can use **SQLPutData** to send parts of data when sending character *C* data to a column with a character, binary, or data source-specific data type or when **SQLPutData** sends binary *C* data to a column with a character, binary, or data source-specific data type. If **SQLPutData** is called more than once under any other conditions, it returns **SQL\_ERROR** and **SQLSTATE 22003** (Numeric value out of range).

## Code Example

In the following example, an application prepares an SQL statement to insert data into the **EMPLOYEE** table. The statement contains parameters for the **NAME**, **ID**, and **PHOTO** columns. For each parameter, the application calls **SQLBindParameter** to specify the C and SQL data types of the parameter. It also specifies that the data for the first and third parameters passes at execution time and that the values 1 and 3 pass for later retrieval by **SQLParamData**. These values identify which parameter is being processed.

The application calls **GetNextID** to get the next available employee ID number. It then calls **SQLExecute** to execute the statement. **SQLExecute** returns **SQL\_NEED\_DATA** when it needs data for the first and third parameters. The application calls **SQLParamData** to retrieve the value that it stored with **SQLBindParameter**; it uses this value to determine for which parameter to send data. For each parameter, the application calls **InitUserData** to initialize the data routine. It repeatedly calls **GetUserData** and **SQLPutData** to get and send the parameter data. Finally, it calls **SQLParamData** to indicate that it has sent all the data for the parameter and to retrieve the value for the next parameter. After data has been sent for both parameters, **SQLParamData** returns **SQL\_SUCCESS**.

For the first parameter, **InitUserData** does nothing, and **GetUserData** calls a routine to prompt the user for the employee name. For the third parameter, **InitUserData** calls a routine to prompt the user for the name of a file containing a bitmap photo of the employee and opens the file. **GetUserData** retrieves the next **MAX\_DATA\_LEN** bytes of photo data from the file. After it retrieves all the photo data, it closes the photo file.

Some application routines are omitted for clarity.

```
#define MAX_DATA_LEN 1024
SDWORD  cbNameParam, cbID = 0; cbPhotoParam, cbData;
SWORD   sID;
PTR     pToken, InitValue;
UCHAR   Data[MAX_DATA_LEN];

retcode = SQLPrepare(hstmt,
    "INSERT INTO EMPLOYEE (NAME, ID, PHOTO) VALUES (?, ?, ?)",
    SQL_NTS);
if (retcode == SQL_SUCCESS) {

    /* Bind the parameters. For parameters 1 and 3, pass the      */
    /* parameter number in rgbValue instead of a buffer address. */
}
```

```

SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
                 NAME_LEN, 0, 1, 0, &cbNameParam);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT,
                 SQL_SMALLINT, 0, 0, &sID, 0, &cbID);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT,
                 SQL_C_BINARY, SQL_LONGVARBINARY,
                 0, 0, 3, 0, &cbPhotoParam);

/* Set values so data for parameters 1 and 3 will be passed */
/* at execution. Note that the length parameter in the macro */
/* SQL_LEN_DATA_AT_EXEC is 0. This assumes that the driver */
/* returns "N" for the SQL_NEED_LONG_DATA_LEN information */
/* type in SQLGetInfo. */

cbNameParam = cbPhotoParam = SQL_LEN_DATA_AT_EXEC(0);

sID = GetNextID(); /* Get next available employee ID number. */

retcode = SQLExecute(hstmt);

/* For data-at-execution parameters, call SQLParamData to get the */
/* parameter number set by SQLBindParameter. Call InitUserData. */
/* Call GetUserData and SQLPutData repeatedly to get and put all */
/* data for the parameter. Call SQLParamData to finish processing */
/* this parameter and start processing the next parameter. */

while (retcode == SQL_NEED_DATA) {
    retcode = SQLParamData(hstmt, &pToken);
    if (retcode == SQL_NEED_DATA) {
        InitUserData((SWORD)pToken, InitValue);
        while (GetUserData(InitValue, (SWORD)pToken, Data, &cbData))
            SQLPutData(hstmt, Data, cbData);
    }
}

VOID InitUserData(sParam, InitValue)
SWORD sParam;
PTR InitValue;
{
    UCHAR szPhotoFile[MAX_FILE_NAME_LEN];
    switch sParam {
        case 3:

            /* Prompt user for bitmap file containing employee photo. */
            /* OpenPhotoFile opens the file and returns the file handle. */

            PromptPhotoFileName(szPhotoFile);
            OpenPhotoFile(szPhotoFile, (FILE *)InitValue);
            break;
    }
}

BOOL GetUserData(InitValue, sParam, Data, cbData)
PTR InitValue;

```



```

SWORD sParam;
UCHAR *Data;
SDWORD *cbData;

{
switch sParam {
case 1:
    /* Prompt user for employee name. */

    PromptEmployeeName(Data);
    *cbData = SQL_NTS;
    return (TRUE);

case 3:
    /* GetNextPhotoData returns the next piece of photo data and */
    /* the number of bytes of data returned (up to MAX_DATA_LEN). */

    Done = GetNextPhotoData((FILE *)InitValue, Data,
                            MAX_DATA_LEN, &cbData);

    if (Done) {
        ClosePhotoFile((FILE *)InitValue);
        return (TRUE);
    }
    return (FALSE);
}
return (FALSE);
}

```

## Related Functions

For information about	See
Canceling statement processing	<b>SQLCancel</b>
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Returning the next parameter to send data for	<b>SQLParamData</b>
Assigning storage for a parameter	<b>SQLBindParameter</b>

## SQLRowCount

**SQLRowCount** returns the number of rows affected by an UPDATE, INSERT, or DELETE statement.

### Syntax

```
RETCODE SQLRowCount(hstmt, pcrow)
```

The **SQLRowCount** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
SDWORD FAR *	<i>pcrow</i>	Output	<p>For UPDATE, INSERT, and DELETE statements, <i>pcrow</i> is the number of rows affected by the request or -1 if the number of affected rows is not available.</p> <p>For other statements and functions, the driver can define the value of <i>pcrow</i>. For example, some data sources might be able to return the number of rows returned by a SELECT statement or a catalog function before fetching the rows.</p> <p>Many data sources cannot return the number of rows in a result set before fetching them; for maximum interoperability, applications should not rely on this behavior.</p>

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLRowCount** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLRowCount** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1010	Function-sequence error	(DM) The function was called prior to calling <b>SQLExecute</b> or <b>SQLExecDirect</b> for the <i>hstmt</i> .  (DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.

## Usage

If the last executed statement associated with *hstmt* was not an UPDATE, INSERT, or DELETE statement, the value of *pcrow* is driver defined.

## Related Functions

For information about	See
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>

## SQLSetConnectOption

**SQLSetConnectOption** sets options that govern aspects of connections.

Level 1

### Syntax

```
RETCODE SQLSetConnectOption(hdbc, fOption, vParam)
```

The **SQLSetConnectOption** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
UWORD	<i>fOption</i>	Input	Option to set, listed in “Usage.”
UDWORD	<i>vParam</i>	Input	Value associated with <i>fOption</i> . Depending on the value of <i>fOption</i> , <i>vParam</i> will be a 32-bit integer value or point to a null-terminated character string.

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLSetConnectOption** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLSetConnectOption** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

The driver can return `SQL_SUCCESS_WITH_INFO` to provide information about the result of setting an option. For example, setting `SQL_ACCESS_MODE` to read only during a transaction might cause the transaction to be committed. The driver could use `SQL_SUCCESS_WITH_INFO`, and information returned with **SQLError**, to inform the application of the commit action.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01S02	Option value changed	The driver did not support the specified value of the <i>vParam</i> argument and substituted a similar value (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
08002	Connection in use	The argument <i>fOption</i> was <code>SQL_ODBC_CURSORS</code> , and the driver was already connected to the data source.
08003	Connection not open	An <i>fOption</i> value was specified that required an open connection, but the <i>hdbc</i> was not in a connected state.
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
IM009	Unable to load translation shared library	The driver was unable to load the translation shared library that was specified for the connection. This error can only be returned when <i>fOption</i> is <code>SQL_TRANSLATE_DLL</code> .

(1 of 2)

SQLSTATE	Error	Description
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	Given the specified <i>fOption</i> value, an invalid value was specified for the argument <i>vParam</i> . (The Driver Manager returns this SQLSTATE only for connection and statement options that accept a discrete set of values, such as SQL_ACCESS_MODE. For all other connection and statement options, the driver must verify the value of the argument <i>vParam</i> .)
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for an <i>hstmt</i> associated with the <i>hdbc</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.  (DM) <b>SQLBrowseConnect</b> was called for the <i>hdbc</i> and returned SQL_NEED_DATA. This function was called before <b>SQLBrowseConnect</b> returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
S1011	Operation invalid at this time	The argument <i>fOption</i> was SQL_TXN_ISOLATION, and a transaction was open.
S1092	Option type out of range	(DM) The value specified for the argument <i>fOption</i> was in the block of numbers reserved for ODBC connection and statement options, but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The driver or data source does not support the value specified for the argument <i>fOption</i> .

(2 of 2)

When *fOption* is a statement option, **SQLSetConnectOption** can return any SQLSTATE that is returned by **SQLSetStmtOption**.

## Usage

The currently defined options are shown in the following table. Options from 0 to 999 are reserved by ODBC.

An application can call **SQLSetConnectOption** and include a statement option. The driver sets the statement option for any *hstmts* associated with the specified *hdbc* and establishes the statement option as a default for any *hstmts* later allocated for that *hdbc*. For a list of statement options, see [“Usage” on page 13-288](#).

All connection and statement options that the application successfully sets for the *hdbc* persist until **SQLFreeConnect** is called for the *hdbc*. For example, if an application calls **SQLSetConnectOption** before it connects to a data source, the option persists even if **SQLSetConnectOption** fails in the driver when the application connects to the data source; if an application sets a driver-specific option, the option persists even if the application connects to a different driver on the *hdbc*.

Some connection and statement options support substituting a similar value if the data source does not support the specified value of *vParam*. In such cases, the driver returns SQL\_SUCCESS\_WITH\_INFO and SQLSTATE 01S02 (Option value changed). For example, if *fOption* is SQL\_PACKET\_SIZE and *vParam* exceeds the maximum packet size, the driver substitutes the maximum size. To determine the substituted value, an application calls **SQLGetConnectOption** (for connection options) or **SQLGetStmtOption** (for statement options).

The format of information set through *vParam* depends on the specified *fOption*. **SQLSetConnectOption** accepts option information in one of two formats: a null-terminated character string or a 32-bit integer value. The format of each *fOption* is noted in the following table. Character strings pointed to by the *vParam* argument of **SQLSetConnectOption** have a maximum length of SQL\_MAX\_OPTION\_STRING\_LENGTH bytes (excluding the null-termination byte).

<i>fOption</i>	<i>vParam Contents</i>
SQL_ACCESS_MODE	A 32-bit integer value. SQL_MODE_READ_ONLY is used by the driver or data source as an indicator that the connection is not required to support SQL statements that cause updates to occur. This mode can be used to optimize locking strategies, transaction management, or other areas as appropriate to the driver or data source. The driver is not required to prevent such statements from being submitted to the data source. The behavior of the driver and data source when asked to process SQL statements that are not read only during a read-only connection is implementation defined. SQL_MODE_READ_WRITE is the default.
SQL_AUTOCOMMIT	A 32-bit integer value that specifies whether to use autocommit or manual-commit mode:  SQL_AUTOCOMMIT_OFF = The driver uses manual-commit mode, and the application must explicitly commit or roll back transactions with <b>SQLTransact</b> .  SQL_AUTOCOMMIT_ON = The driver uses autocommit mode. Each statement is committed immediately after it is executed. This is the default. Changing from manual-commit mode to autocommit mode commits any open transactions on the connection.  <b>Important:</b> Some data sources delete the access plans and close the cursors for all <i>hstmts</i> on an <i>hdbc</i> each time a statement is committed; autocommit mode can cause this to happen after each statement is executed. For more information, see the “ <a href="#">SQL_CURSOR_COMMIT_BEHAVIOR</a> ” and “ <a href="#">SQL_CURSOR_ROLLBACK_BEHAVIOR</a> ” on page 13-192.
SQL_CURRENT_QUALIFIER UNSUPPORTED	INFORMIX-CLI does not support this option.
SQL_LOGIN_TIMEOUT UNSUPPORTED	INFORMIX-CLI does not support this option.

(1 of 4)



<i>fOption</i>	<i>vParam Contents</i>
SQL_ODBC_CURSORS	<p>A 32-bit option specifying how the driver manager uses the ODBC cursor library:</p> <p>SQL_CUR_USE_IF_NEEDED = The driver manager uses the ODBC cursor library only if it is needed. If the driver supports the SQL_FETCH_PRIOR option in <b>SQLExtendedFetch</b>, the driver manager uses the scrolling capabilities of the driver. Otherwise, it uses the ODBC cursor library.</p> <p>SQL_CUR_USE_ODBC = The driver manager uses the ODBC cursor library.</p> <p>SQL_CUR_USE_DRIVER = The driver manager uses the scrolling capabilities of the driver. This is the default setting.</p> <p>Informix recommends that you use SQL_CUR_USE_DRIVER. For more information about the ODBC cursor library, see the <i>Microsoft ODBC Programmer's Reference and SDK Guide</i>, Version 2.0.</p>
SQL_OPT_TRACE	<p>A 32-bit integer value telling the driver manager whether to perform tracing:</p> <p>SQL_OPT_TRACE_OFF = Tracing off (the default)</p> <p>SQL_OPT_TRACE_ON = Tracing on</p> <p>When tracing is on, the driver manager writes each INFORMIX-CLI function call to the trace file.</p> <p>When tracing is on, the driver manager can return SQLSTATE IM013 (Trace-file error) from any function.</p> <p>An application specifies a trace file with the SQL_OPT_TRACEFILE option. If the file already exists, the driver manager appends to the file. Otherwise, it creates the file. If tracing is on but no trace file has been specified, the driver manager writes to the file <b>sql.log</b> in the current directory.</p> <p>If the Trace keyword in the ODBC section of the <b>odbc.ini</b> file is set to 1 when an application calls <b>SQLAllocEnv</b>, tracing is enabled.</p>
SQL_OPT_TRACEFILE	<p>A null-terminated character string containing the name of the trace file.</p> <p>The default value of the SQL_OPT_TRACEFILE option is specified with the TraceFile keyname in the ODBC section of the <b>odbc.ini</b> file. For more information, see <a href="#">“ODBC Options” on page 1-10</a>.</p>
SQL_PACKET_SIZE	<p>A 32-bit integer value specifying the network packet size in bytes.</p> <p>Many data sources either do not support this option or can only return the network packet size.</p> <p>If the specified size exceeds the maximum packet size or is smaller than the minimum packet size, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).</p>

<i>fOption</i>	<i>vParam</i> Contents
SQL_QUIET_MODE	<p>A 32-bit window handle (<i>hwnd</i>).</p> <p>If the window handle is a null pointer, the driver does not display any dialog boxes.</p> <p>If the window handle is not a null pointer, it should be the parent window handle of the application. The driver uses this handle to display dialog boxes. This is the default.</p> <p>If the application has not specified a parent window handle for this option, the driver uses a null parent window handle to display dialog boxes or return in <b>SQLGetConnectOption</b>.</p> <p>The SQL_QUIET_MODE connection option does not apply to dialog boxes displayed by <b>SQLDriverConnect</b>.</p>
SQL_TRANSLATE_DLL	<p>A null-terminated character string that contains the name of a shared library that contains the functions <b>SQLDriverToDataSource</b> and <b>SQLDataSourceToDriver</b> that the driver loads and uses to perform tasks such as character set translation. This option can be specified only if the driver has connected to the data source. For more information about translating data, see <a href="#">Chapter 15, “Translation Shared Library Function Reference.”</a></p>
SQL_TRANSLATE_OPTION	<p>A 32-bit flag value that is passed to the translation shared library. This option can only be specified if the driver has connected to the data source.</p>

(3 of 4)

<i>fOption</i>	<i>vParam</i> Contents
SQL_TXN_ISOLATION	<p>A 32-bit mask that sets the transaction isolation level for the current <i>hdbc</i>. An application must call <b>SQLTransact</b> to commit or roll back all open transactions on an <i>hdbc</i> before calling <b>SQLSetConnectOption</b> with this option.</p> <p>The valid values for <i>vParam</i> can be determined by calling <b>SQLGetInfo</b> with <i>fInfoType</i> equal to SQL_TXN_ISOLATION_OPTIONS. The following terms are used to define transaction isolation levels:</p> <p><i>Dirty Read</i>: Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits this change. Transaction 1 rolls back the change, and transaction 2 reads a row that is considered to have never existed.</p> <p><i>Nonrepeatable Read</i>: Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. Transaction 1 attempts to reread the row, and it receives different row values or discovers that the row has been deleted.</p> <p><i>Phantom</i>: Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. Transaction 1 reexecutes the statement that read the rows, and it receives a different set of rows.</p> <p><i>vParam</i> must be one of the following values:</p> <p>SQL_TXN_READ_UNCOMMITTED = Dirty Reads, Nonrepeatable Reads, and Phantoms are possible.</p> <p>SQL_TXN_READ_COMMITTED = Dirty Reads are not possible. Nonrepeatable Reads and Phantoms are possible.</p> <p>SQL_TXN_REPEATABLE_READ = Dirty Reads and Nonrepeatable Reads are not possible. Phantoms are possible.</p> <p>SQL_TXN_SERIALIZABLE = Transactions are serializable. Dirty Reads, Nonrepeatable Reads, and Phantoms are not possible.</p> <p>SQL_TXN_VERSIONING = Transactions are serializable, but higher concurrency is possible than with SQL_TXN_SERIALIZABLE. Dirty Reads are not possible. Typically, SQL_TXN_SERIALIZABLE is implemented by using locking protocols that reduce concurrency, and SQL_TXN_VERSIONING is implemented by using nonlocking protocol such as record versioning.</p>

(4 of 4)

## Data Translation

Data translation is performed for all data moving between the driver and the data source.

The translation option (set with the `SQL_TRANSLATE_OPTION` option) can be any 32-bit value. Its meaning depends on the translation shared library that you use. A new option can be set at any time and will be applied to the next exchange of data following a call to **SQLSetConnectOption**. A default translation shared library can be specified for the data source in its data-source specification in the **odbc.ini** file. The default translation shared library is loaded by the driver at connection time. A translation option (`SQL_TRANSLATE_OPTION`) can also be specified in the data-source specification.

To change the translation shared library for a connection, an application calls **SQLSetConnectOption** with the `SQL_TRANSLATE_DLL` option after it connects to the data source. The driver attempts to load the specified shared library and, if the attempt fails, the driver returns `SQL_ERROR` with the `SQLSTATE IM009` (Unable to load translation shared library).

If no translation shared library has been specified in the ODBC initialization file or by calling **SQLSetConnectOption**, the driver does not attempt to translate data. Any value set for the translation option is ignored.

For more information about translating data, see [“ODBC Options” on page 1-10](#) and [Chapter 15, “Translation Shared Library Function Reference.”](#)

## Code Example

See **SQLConnect**.

## Related Functions

For information about	See
Returning the setting of a connection option	<b>SQLGetConnectOption</b>
Returning the setting of a statement option	<b>SQLGetStmtOption</b>
Setting a statement option	<b>SQLSetStmtOption</b>

## Core

## SQLSetCursorName

**SQLSetCursorName** associates a cursor name with an active *hstmt*. If an application does not call **SQLSetCursorName**, the driver generates cursor names as needed for SQL statement processing.

### Syntax

```
RETCODE SQLSetCursorName(hstmt, szCursor, cbCursor)
```

The **SQLSetCursorName** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
UCHAR FAR *	<i>szCursor</i>	Input	Cursor name
SWORD	<i>cbCursor</i>	Input	Length of <i>szCursor</i>

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLSetCursorName** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLSetCursorName** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
24000	Invalid cursor state	The statement corresponding to <i>hstmt</i> was already in an executed or cursor-positioned state.
34000	Invalid cursor name	The cursor name specified by the argument <i>szCursor</i> was invalid. For example, the cursor name exceeded the maximum length as defined by the driver.
3C000	Duplicate cursor name	The cursor name specified by the argument <i>szCursor</i> already exists.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

(1 of 2)

SQLSTATE	Error	Description
S1009	Invalid argument value	(DM) The argument <i>szCursor</i> was a null pointer.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The argument <i>cbCursor</i> was less than 0, but not equal to SQL_NTS.

(2 of 2)

## Usage

The only ODBC SQL statements that use a cursor name are a positioned UPDATE and DELETE (for example, UPDATE *table-name* ... WHERE CURRENT OF *cursor-name*). If the application does not call **SQLSetCursorName** to define a cursor name, when a SELECT statement executes, the driver generates a name that begins with the letters SQL\_CUR and does not exceed 18 characters.

All cursor names within the *hdbc* must be unique. The maximum length of a cursor name is defined by the driver. For maximum interoperability, cursor names should not exceed 18 characters.

A cursor name remains set until the *hstmt* with which it is associated is dropped, using **SQLFreeStmt** with the SQL\_DROP option.

## Code Example

In the following example, an application uses **SQLSetCursorName** to set a cursor name for an *hstmt*. It then uses that *hstmt* to retrieve results from the **EMPLOYEE** table. Finally, it performs a positioned update to change the name of 25-year-old John Smith to John D. Smith. The application uses different *hstmts* for the SELECT and UPDATE statements.

```
#define NAME_LEN 30

HSTMT      hstmtSelect,
HSTMT      hstmtUpdate;
UCHAR      szName[NAME_LEN];
SWORD      sAge;
SDWORD     cbName;
SDWORD     cbAge;

/* Allocate the statements and set the cursor name */

SQLAllocStmt(hdbc, &hstmtSelect);
SQLAllocStmt(hdbc, &hstmtUpdate);
SQLSetCursorName(hstmtSelect, "C1", SQL_NTS);

/* SELECT the result set and bind its columns to local storage */

SQLExecDirect(hstmtSelect,
              "SELECT NAME, AGE FROM EMPLOYEE FOR UPDATE",
              SQL_NTS);
SQLBindCol(hstmtSelect, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
SQLBindCol(hstmtSelect, 2, SQL_C_SSHORT, &sAge, 0, &cbAge);

/* Read through the result set until the cursor is      */
/* positioned on the row for the 25-year-old John Smith */

do
    retcode = SQLFetch(hstmtSelect);
while ((retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) &&
      (strcmp(szName, "Smith, John") != 0 || sAge != 25));

/* Perform a positioned update of John Smith's name */

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
    SQLExecDirect(hstmtUpdate,
                  "UPDATE EMPLOYEE SET NAME=\"Smith, John D.\" WHERE CURRENT OF C1",
                  SQL_NTS);
}
```



## Related Functions

For information about	See
Executing an SQL statement	<b>SQLExecDirect</b>
Executing a prepared SQL statement	<b>SQLExecute</b>
Returning a cursor name	<b>SQLGetCursorName</b>
Setting cursor scrolling options	<b>SQLSetScrollOptions</b>

## SQLSetParam

### Deprecated

**SQLBindParameter** replaces the ODBC 1.0 function **SQLSetParam**. For more information, see **SQLBindParameter**.

## SQLSetScrollOptions

In ODBC 2.0, the `SQL_CURSOR_TYPE`, `SQL_CONCURRENCY`, `SQL_KEYSET_SIZE`, and `SQL_ROWSET_SIZE` options for **SQLSetStmtOption** superseded **SQLSetScrollOptions**. INFORMIX-CLI 2.5 applications should not call this function.

Level 2

---

## SQLSetStmtOption

**SQLSetStmtOption** sets options that are related to an *hstmt*. To set an option for all the statements associated with a specific *hdbc*, an application can call **SQLSetConnectOption**.

### Syntax

```
RETCODE SQLSetStmtOption(hstmt, fOption, vParam)
```

The **SQLSetStmtOption** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UWORD	<i>fOption</i>	Input	Option to set, listed in “Usage.”
UDWORD	<i>vParam</i>	Input	Value associated with <i>fOption</i> . Depending on the value of <i>fOption</i> , <i>vParam</i> will be a 32-bit integer value or point to a null-terminated character string.

---

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLSetStmtOption** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLSetStmtOption** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01S02	Option value changed	The driver did not support the specified value of the <i>vParam</i> argument and substituted a similar value (function returns <code>SQL_SUCCESS_WITH_INFO</code> ).
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	The <i>fOption</i> was <code>SQL_CONCURRENCY</code> , <code>SQL_SIMULATE_CURSOR</code> , or <code>SQL_CURSOR_TYPE</code> , and the cursor was open.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	Given the specified <i>fOption</i> value, an invalid value was specified for the argument <i>vParam</i> . (The driver manager returns this <code>SQLSTATE</code> only for statement options that accept a discrete set of values, such as <code>SQL_ASYNC_ENABLE</code> . For all other statement options, the driver must verify the value of the argument <i>vParam</i> .)

(1 of 2)

SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1011	Operation invalid at this time	The <i>fOption</i> was SQL_CONCURRENCY, SQL_SIMULATE_CURSOR, or SQL_CURSOR_TYPE, and the statement was prepared.
S1092	Option type out of range	(DM) The value specified for the argument <i>fOption</i> was in the block of numbers reserved for ODBC connection and statement options, but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The driver or data source does not support the value specified for the argument <i>fOption</i> .

(2 of 2)

## Usage

Statement options for an *hstmt* remain in effect until they are changed by another call to **SQLSetStmtOption** or the *hstmt* is dropped by calling **SQLFreeStmt** with the SQL\_DROP option. Calling **SQLFreeStmt** with the SQL\_CLOSE, SQL\_UNBIND, or SQL\_RESET\_PARAMS options does not reset statement options.

Some statement options support substituting a similar value if the data source does not support the specified value of *vParam*. In such cases, the driver returns SQL\_SUCCESS\_WITH\_INFO and SQLSTATE 01S02 (Option value changed). For example, if *fOption* is SQL\_CONCURRENCY, *vParam* is SQL\_CONCUR\_ROWVER, and the data source does not support this, the driver substitutes SQL\_CONCUR\_VALUES. To determine the substituted value, an application calls **SQLGetStmtOption**.

The currently defined options are shown in the following table. Options from 0 to 999 are reserved by ODBC.

The format of information set with *vParam* depends on the specified *fOption*. **SQLSetStmtOption** accepts option information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format is noted in the option description. This format applies to the information returned for each option in **SQLGetStmtOption**. Character strings pointed to by the *vParam* argument of **SQLSetStmtOption** have a maximum length of `SQL_MAX_OPTION_STRING_LENGTH` bytes (excluding the null-termination byte).

<i>fOption</i>	<i>vParam</i> Contents
SQL_ASYNC_ENABLE UNSUPPORTED	INFORMIX-CLI does not support this option. The default is set to <code>SQL_ASYNC_ENABLE_OFF</code> .
SQL_BIND_TYPE	<p>A 32-bit integer value that sets the binding orientation to be used when <b>SQLExtendedFetch</b> is called on the associated <i>hstmt</i>. Column-wise binding is selected by supplying the defined constant <code>SQL_BIND_BY_COLUMN</code> for the argument <i>vParam</i>. Row-wise binding is selected by supplying a value for <i>vParam</i> specifying the length of a structure or an instance of a buffer into which result columns will be bound.</p> <p>The length specified in <i>vParam</i> 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 will point to the beginning of the same column in the next row. When using the <b>sizeof</b> operator with structures or unions in ANSI C, this behavior is guaranteed.</p> <p>Column-wise binding is the default binding orientation for <b>SQLExtendedFetch</b>.</p>

(1 of 5)

<i>fOption</i>	<i>vParam</i> Contents
SQL_CONCURRENCY	<p data-bbox="384 256 1013 280">A 32-bit integer value that specifies the cursor concurrency:</p> <p data-bbox="384 315 1170 341">SQL_CONCUR_READ_ONLY = Cursor is read only. No updates are allowed.</p> <p data-bbox="384 342 1170 396">SQL_CONCUR_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated.</p> <p data-bbox="384 397 1123 451">SQL_CONCUR_ROWVER = Cursor uses optimistic concurrency control, comparing row versions.</p> <p data-bbox="384 453 1112 506">SQL_CONCUR_VALUES = Cursor uses optimistic concurrency control, comparing values.</p> <p data-bbox="384 537 1177 618">The default value is SQL_CONCUR_READ_ONLY. This option can also be set through the <i>fConcurrency</i> argument in <b>SQLSetScrollOptions</b>. This option cannot be specified for an open cursor.</p> <p data-bbox="384 649 1208 756">If the SQL_CURSOR_TYPE <i>fOption</i> is changed to a type that does not support the current value of SQL_CONCURRENCY, the value of SQL_CONCURRENCY is not automatically changed to a supported value, and no error will be reported until <b>SQLExecDirect</b> or <b>SQLPrepare</b> is called.</p> <p data-bbox="384 787 1193 1008">If the driver supports the SELECT_FOR_UPDATE statement, and such a statement is executed while the value of SQL_CONCURRENCY is set to SQL_CONCUR_READ_ONLY, an error will be returned. If the value of SQL_CONCURRENCY is changed to a value that the driver supports for some value of SQL_CURSOR_TYPE, but not for the current value of SQL_CURSOR_TYPE, the value of SQL_CURSOR_TYPE is not automatically changed to a supported value, and no error will be reported until <b>SQLExecDirect</b> or <b>SQLPrepare</b> is called.</p> <p data-bbox="384 1047 1208 1187">If the specified concurrency is not supported by the data source, the driver substitutes a different concurrency and returns SQLSTATE 01S02 (Option value changed). For SQL_CONCUR_VALUES, the driver substitutes SQL_CONCUR_ROWVER, and vice versa. For SQL_CONCUR_LOCK, the driver substitutes, in order, SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES.</p>

(2 of 5)

<i>fOption</i>	<i>vParam Contents</i>
SQL_CURSOR_TYPE	<p>A 32-bit integer value that specifies the cursor type:</p> <p>SQL_CURSOR_FORWARD_ONLY = The cursor only scrolls forward.</p> <p>SQL_CURSOR_STATIC = The data in the result set is static.</p> <p>SQL_CURSOR_KEYSET_DRIVEN = The driver saves and uses the keys for the number of rows specified in the SQL_KEYSET_SIZE statement option.</p> <p>SQL_CURSOR_DYNAMIC = The driver only saves and uses the keys for the rows in the rowset.</p> <p>The default value is SQL_CURSOR_FORWARD_ONLY. This option cannot be specified for an open cursor and can also be set through the <i>crowKeyset</i> argument in <b>SQLSetScrollOptions</b>.</p> <p>If the specified cursor type is not supported by the data source, the driver substitutes a different cursor type and returns SQLSTATE 01S02 (Option value changed). For a mixed or dynamic cursor, the driver substitutes, in order, a keyset-driven or static cursor. For a keyset-driven cursor, the driver substitutes a static cursor.</p>
SQL_KEYSET_SIZE	<p>A 32-bit integer value that specifies the number of rows in the keyset for a keyset-driven cursor. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside of the keyset). The default keyset size is 0.</p> <p>If the specified size exceeds the maximum keyset size, the driver substitutes that size and returns SQLSTATE 01S02 (Option value changed).</p> <p><b>SQLExtendedFetch</b> returns an error if the keyset size is greater than 0 and less than the rowset size.</p>

(3 of 5)

<i>fOption</i>	<i>vParam</i> Contents
SQL_MAX_LENGTH	<p>A 32-bit integer value that specifies the maximum amount of data that the driver returns from a character or binary column. If <i>vParam</i> is less than the length of the available data, <b>SQLFetch</b> or <b>SQLGetData</b> truncates the data and returns SQL_SUCCESS. If <i>vParam</i> is 0 (the default), the driver attempts to return all available data.</p> <p>If the specified length is less than the minimum amount of data that the data source can return (the minimum is 254 bytes on many data sources), or greater than the maximum amount of data that the data source can return, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).</p> <p>This option is intended to reduce network traffic and should only be supported when the data source (as opposed to the driver) in a multiple-tier driver can implement it. To truncate data, an application should specify the maximum buffer length in the <i>cbValueMax</i> argument in <b>SQLBindCol</b> or <b>SQLGetData</b>.</p> <p>In ODBC 1.0, this statement option only applied to SQL_LONGVARCHAR and SQL_LONGVARBINARY columns.</p>
SQL_MAX_ROWS	<p>A 32-bit integer value corresponding to the maximum number of rows to return to the application for a SELECT statement. If <i>vParam</i> equals 0 (the default), then the driver returns all rows.</p> <p>This option is intended to reduce network traffic. Conceptually, it is applied when the result set is created and limits the result set to the first <i>vParam</i> rows.</p> <p>If the specified number of rows exceeds the number of rows that can be returned by the data source, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).</p>
SQL_NOSCAN	<p>A 32-bit integer value that specifies whether the driver does not scan SQL strings for escape clauses:</p> <p>SQL_NOSCAN_OFF = The driver scans SQL strings for escape clauses (the default).</p> <p>SQL_NOSCAN_ON = The driver does not scan SQL strings for escape clauses. Instead, the driver sends the statement directly to the data source.</p>
SQL_QUERY_TIMEOUT UNSUPPORTED	<p>INFORMIX-CLI does not support this option. The default is set to 0 (no time-out).</p>



<i>fOption</i>	<i>vParam Contents</i>
SQL_RETRIEVE_DATA	<p>A 32-bit integer value:</p> <p>SQL_RD_ON = <b>SQLExtendedFetch</b> retrieves data after it positions the cursor to the specified location. This is the default.</p> <p>SQL_RD_OFF = <b>SQLExtendedFetch</b> does not retrieve data after it positions the cursor.</p> <p>By setting SQL_RETRIEVE_DATA to SQL_RD_OFF, an application can verify if a row exists without incurring the overhead of retrieving rows.</p>
SQL_ROWSET_SIZE	<p>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 <b>SQLExtendedFetch</b>. The default value is 1.</p> <p>If the specified rowset size exceeds the maximum rowset size supported by the data source, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).</p> <p>This option can be specified for an open cursor and can also be set through the <i>crowRowset</i> argument in <b>SQLSetScrollOptions</b>.</p>
SQL_SIMULATE_CURSOR <b>UNSUPPORTED</b>	INFORMIX-CLI does not support this option.
SQL_USE_BOOKMARKS <b>UNSUPPORTED</b>	INFORMIX-CLI does not support this option. The default is set to SQL_UB_OFF = Off

(5 of 5)

## Code Example

See **SQLExtendedFetch**.

## Related Functions

---

<b>For information about</b>	<b>See</b>
Canceling statement processing	<b>SQLCancel</b>
Returning the setting of a connection option	<b>SQLGetConnectOption</b>
Returning the setting of a statement option	<b>SQLGetStmtOption</b>
Setting a connection option	<b>SQLSetConnectOption</b>

---

## SQLSpecialColumns

**SQLSpecialColumns** retrieves the following information about columns within a specified table:

- The optimal set of columns that uniquely identifies a row in the table
- Columns that are automatically updated when any value in the row is updated by a transaction

### Syntax

```
RETCODE SQLSpecialColumns(hstmt, fColType, szTableQualifier,
cbTableQualifier, szTableOwner, cbTableOwner, szTableName,
cbTableName, fScope, fNullable)
```

The **SQLSpecialColumns** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle
UWORD	<i>fColType</i>	Input	Type of column to return. Must be one of the following values:  SQL_BEST_ROWID: Returns the optimal column or set of columns that, by retrieving values from the column or columns, allows any row in the specified table to be uniquely identified. A column can be either a pseudo-column specifically designed for this purpose or the column or columns of any unique index for the table.  SQL_ROWVER: Returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction.

Type	Argument	Use	Description
UCHAR FAR *	<i>szTableQualifier</i>	Input	Qualifier name for the table. If a driver supports qualifiers for some tables but not for others, as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i>
UCHAR FAR *	<i>szTableOwner</i>	Input	Owner name for the table. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i>
UCHAR FAR *	<i>szTableName</i>	Input	Table name
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i>

(2 of 3)

Type	Argument	Use	Description
UWORD	<i>fScope</i>	Input	<p>Minimum required scope of the rowid. The returned rowid might be of greater scope. It must be one of the following:</p> <p>SQL_SCOPE_CURROW: The rowid is guaranteed to be valid only while positioned on that row. A later reselect using rowid might not return a row if the row was updated or deleted by another transaction.</p> <p>SQL_SCOPE_TRANSACTION: The rowid is guaranteed to be valid for the duration of the current transaction.</p> <p>SQL_SCOPE_SESSION: The rowid is guaranteed to be valid for the duration of the session (across transaction boundaries).</p>
UWORD	<i>fNullable</i>	Input	<p>Determines whether to return special columns that can have a NULL value. It must be one of the following:</p> <p>SQL_NO_NULLS: Exclude special columns that can have NULL values.</p> <p>SQL_NULLABLE: Return special columns even if they can have NULL values.</p>

(3 of 3)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLSpecialColumns** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLSpecialColumns** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.

(1 of 2)

SQLSTATE	Error	Description
S1090	Invalid string or buffer length	<p>(DM) 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 value for the corresponding qualifier or name. The maximum length of each qualifier or name can be obtained by calling <b>SQLGetInfo</b> with the <i>fInfoType</i> values:</p> <p>SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, or SQL_MAX_TABLE_NAME_LEN</p>
S1097	Column type out of range	(DM) An invalid <i>fColType</i> value was specified.
S1098	Scope type out of range	(DM) An invalid <i>fScope</i> value was specified.
S1099	Nullable type out of range	(DM) An invalid <i>fNullable</i> value was specified.
S1C00	Driver not capable	<p>A table qualifier was specified, but the driver or data source does not support qualifiers.</p> <p>A table owner was specified, but the driver or data source does not support owners.</p> <p>The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.</p>
S1T00	Time-out expired	<p>The time-out period expired before the data source returned the requested result set. The time-out period is set through <b>SQLSetStmtOption</b>, SQL_QUERY_TIMEOUT.</p>

(2 of 2)

## Usage

**SQLSpecialColumns** is provided so that applications can provide their own custom scrollable-cursor functionality, similar to the functionality provided by **SQLExtendedFetch** and **SQLSetStmtOption**.

When the *fColType* argument is `SQL_BEST_ROWID`, **SQLSpecialColumns** returns the column or columns that uniquely identify each row in the table. These columns can always be used in a `SELECT` list or `WHERE` clause. However, **SQLColumns** does not necessarily return these columns. If no columns uniquely identify each row in the table, **SQLSpecialColumns** returns a rowset with no rows; a subsequent call to **SQLFetch** or **SQLExtendedFetch** on the *hstmt* returns `SQL_NO_DATA_FOUND`.

If the *fColType*, *fScope*, or *fNullable* arguments specify characteristics that are not supported by the data source, **SQLSpecialColumns** returns a result set with no rows (as opposed to the function returning `SQL_ERROR` with `SQLSTATE S1C00` (Driver not capable)). A subsequent call to **SQLFetch** or **SQLExtendedFetch** on the *hstmt* returns `SQL_NO_DATA_FOUND`.

**SQLSpecialColumns** returns the results as a standard result set, ordered by `SCOPE`. The following table lists the columns in the result set.



The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual length of the COLUMN\_NAME column, an application can call **SQLGetInfo** with the SQL\_MAX\_COLUMN\_NAME\_LEN option.

Column Name	Data Type	Comments
SCOPE	SMALLINT	Actual scope of the rowid. Contains one of the following values:  SQL_SCOPE_CURROW SQL_SCOPE_TRANSACTION SQL_SCOPE_SESSION  NULL is returned when <i>fColType</i> is SQL_ROWVER.  For a description of each value, see the description of “fScope” on page 13-297.
COLUMN_NAME	VARCHAR(128) not NULL	Column identifier.
DATA_TYPE	SMALLINT not NULL	For a list of valid ODBC SQL data types, see <a href="#">Appendix C</a> . For information on how Informix data types map to ODBC SQL data types, see “ <a href="#">Mapping Data Types</a> ” on page 1-15
TYPE_NAME	VARCHAR(128) not NULL	Data source-dependent data-type name; for example, CHAR, VARCHAR, MONEY, LONG VARBINARY, or CHAR ( ) FOR BIT DATA.
PRECISION	INTEGER	The precision of the column on the data source. NULL is returned for data types where precision is not applicable. For more information concerning precision, see “ <a href="#">Precision, Scale, Length, and Display Size</a> ” on page C-8.”

(1 of 2)

Column Name	Data Type	Comments
LENGTH	Integer	The length in bytes of data transferred on an <b>SQLGetData</b> or <b>SQLFetch</b> operation if <b>SQL_C_DEFAULT</b> is specified. For numeric data, this size might be different than the size of the data stored on the data source. This value is the same as the <b>PRECISION</b> column for character or binary data. For more information, see <a href="#">“Precision, Scale, Length, and Display Size”</a> on page C-8.
SCALE	SMALLINT	The scale of the column on the data source. NULL is returned for data types where scale is not applicable. For more information concerning scale, see <a href="#">“Precision, Scale, Length, and Display Size”</a> on page C-8.
PSEUDO_COLUMN	SMALLINT	Returns one of the following values to indicate whether the column is a pseudo-column:  SQL_PC_UNKNOWN SQL_PC_PSEUDO SQL_PC_NOT_PSEUDO

**Important:** For maximum interoperability, pseudo-columns should not be quoted with the identifier quote character returned by **SQLGetInfo**.

(2 of 2)

Once the application retrieves values for **SQL\_BEST\_ROWID**, the application can use these values to reselect that row within the defined scope. The **SELECT** statement is guaranteed to return either no rows or one row.

If an application reselects a row based on the rowid column or columns and the row is not found, the application can assume that the row was deleted or the rowid columns were modified. The opposite is not true: Even if the rowid has not changed, the other columns in the row might have changed.

Columns returned for column type **SQL\_BEST\_ROWID** are useful for applications that need to scroll forward and backward within a result set to retrieve the most recent data from a set of rows. The rowid column or columns are guaranteed not to change while positioned on that row.



The rowid column or columns might remain valid even when the cursor is not positioned on the row; the application can determine this by checking the SCOPE column in the result set.

Columns returned for column type SQL\_ROWVER are useful for applications that need the ability to check if any columns in a given row have been updated while the row was reselected using the rowid. For example, after reselecting a row using rowid, the application can compare the previous values in the SQL\_ROWVER columns to the ones last fetched. If the value in a SQL\_ROWVER column differs from the previous value, the application can alert the user that data on the display has changed.

## Code Example

For a code example of a similar function, see [SQLColumns](#).

## Related Functions

For information about	See
Assigning storage for a column in a result set	<a href="#">SQLBindCol</a>
Canceling statement processing	<a href="#">SQLCancel</a>
Returning the columns in a table or tables	<a href="#">SQLColumns</a>
Fetching a block of data or scrolling through a result set	<a href="#">SQLExtendedFetch</a>
Fetching a row of data	<a href="#">SQLFetch</a>
Returning the columns of a primary key	<a href="#">SQLPrimaryKeys</a>

## SQLStatistics

**SQLStatistics** retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns this information as a result set.

### Syntax

```
RETCODE SQLStatistics(hstmt, szTableQualifier,
    cbTableQualifier, szTableOwner, cbTableOwner, szTableName,
    cbTableName, fUnique, fAccuracy)
```

The **SQLStatistics** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szTableQualifier</i>	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i> .
UCHAR FAR *	<i>szTableOwner</i>	Input	Owner name. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i> .
UCHAR FAR *	<i>szTableName</i>	Input	Table name.
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .

(1 of 2)

Type	Argument	Use	Description
UWORD	<i>fUnique</i>	Input	Type of index:  SQL_INDEX_UNIQUE or SQL_INDEX_ALL
UWORD	<i>fAccuracy</i>	Input	The importance of the CARDI- NALITY and PAGES columns in the result set:  SQL_ENSURE requests that the driver unconditionally retrieve the statistics.  SQL_QUICK requests that the driver retrieve results only if they are readily available from the server. In this case, the driver does not ensure that the values are current.

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING,  
SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLStatistics** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLStatistics** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLERROR</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.  The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name.
S1100	Uniqueness option type out of range	(DM) An invalid <i>fUnique</i> value was specified.

(1 of 2)

SQLSTATE	Error	Description
S1101	Accuracy option type out of range	(DM) An invalid <i>fAccuracy</i> value was specified.
S1C00	Driver not capable	A table qualifier was specified, but the driver or data source does not support qualifiers.  A table owner was specified, but the driver or data source does not support owners.  The combination of the current settings of the <code>SQL_CONCURRENCY</code> and <code>SQL_CURSOR_TYPE</code> statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the requested result set. The time-out period is set through <code>SQLSetStmtOption</code> , <code>SQL_QUERY_TIMEOUT</code> .

(2 of 2)

## Usage

**SQLStatistics** returns information about a single table as a standard result set, ordered by `NON_UNIQUE`, `TYPE`, `INDEX_QUALIFIER`, `INDEX_NAME`, and `SEQ_IN_INDEX`. The result set combines statistics information for the table with information about each index. The following table lists the columns in the result set.

**Important:** *SQLStatistics* might not return all indexes. Applications can use any valid index, regardless of whether it is returned by *SQLStatistics*.



The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE\_QUALIFIER, TABLE\_OWNER, TABLE\_NAME, and COLUMN\_NAME columns, an application can call **SQLGetInfo** with the SQL\_MAX\_QUALIFIER\_NAME\_LEN, SQL\_MAX\_OWNER\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	VARCHAR(128)	Table-qualifier identifier of the table to which the statistic or index applies; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	VARCHAR(128)	Table-owner identifier of the table to which the statistic or index applies; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	VARCHAR(128) not NULL	Table identifier of the table to which the statistic or index applies
NON_UNIQUE	SMALLINT	Indicates whether the index prohibits duplicate values:  TRUE if the index values can be nonunique.  FALSE if the index values must be unique.  NULL is returned if TYPE is SQL_TABLE_STAT.

(1 of 3)



Column Name	Data Type	Comments
INDEX_QUALIFIER	VARCHAR(128)	The identifier that is used to qualify the index name doing a DROP INDEX; NULL is returned if an index qualifier is not supported by the data source or if TYPE is SQL_TABLE_STAT. If a non-null value is returned in this column, it must be used to qualify the index name on a DROP INDEX statement; otherwise, the TABLE_OWNER name should be used to qualify the index name.
INDEX_NAME	VARCHAR(128)	Index identifier; NULL is returned if TYPE is SQL_TABLE_STAT.
TYPE	SMALLINT not NULL	Type of information being returned: SQL_TABLE_STAT indicates a statistic for the table.  SQL_INDEX_CLUSTERED indicates a clustered index.  SQL_INDEX_HASHED indicates a hashed index.  SQL_INDEX_OTHER indicates another type of index.
SEQ_IN_INDEX	SMALLINT	Column-sequence number in index (starting with 1); NULL is returned if TYPE is SQL_TABLE_STAT.
COLUMN_NAME	VARCHAR(128)	Column identifier. If the column is based on an expression, such as SALARY + BENEFITS, the expression is returned; if the expression cannot be determined, an empty string is returned. If the index is a filtered index, each column in the filter condition is returned; this might require more than one row. NULL is returned if TYPE is SQL_TABLE_STAT.

(2 of 3)

Column Name	Data Type	Comments
COLLATION	CHAR(1)	Sort sequence for the column; "A" for ascending; "D" for descending; NULL is returned if column sort sequence is not supported by the data source or if TYPE is SQL_TABLE_STAT.
CARDINALITY	INTEGERS	Cardinality of table or index; number of rows in table if TYPE is SQL_TABLE_STAT; number of unique values in the index if TYPE is not SQL_TABLE_STAT; NULL is returned if the value is not available from the data source.
PAGES	INTEGER	Number of pages used to store the index or table; number of pages for the table if TYPE is SQL_TABLE_STAT; number of pages for the index if TYPE is not SQL_TABLE_STAT; NULL is returned if the value is not available from the data source, or if not applicable to the data source.
FILTER_CONDITION	VARHAR(128)	<p>If the index is a filtered index, this is the filter condition, such as SALARY &gt; 30000; if the filter condition cannot be determined, this is an empty string.</p> <p>NULL if the index is not a filtered index, it cannot be determined whether the index is a filtered index, or TYPE is SQL_TABLE_STAT.</p>

(3 of 3)

If the row in the result set corresponds to a table, the driver sets TYPE to SQL\_TABLE\_STAT and sets NON\_UNIQUE, INDEX\_QUALIFIER, INDEX\_NAME, SEQ\_IN\_INDEX, COLUMN\_NAME, and COLLATION to NULL. If CARDINALITY or PAGES are not available from the data source, the driver sets them to NULL.

## Code Example

For a code example of a similar function, see [SQLColumns](#).

## Related Functions

<b>For information about</b>	<b>See</b>
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning the columns of a primary key	<b>SQLPrimaryKeys</b>

## SQLTablePrivileges

**SQLTablePrivileges** returns a list of tables and the privileges associated with each table. The driver returns the information as a result set on the specified *hstmt*.

### Syntax

```
RETCODE SQLTablePrivileges(hstmt, szTableQualifier,
                           cbTableQualifier,
                           szTableOwner, cbTableOwner, szTableName, cbTableName)
```

The **SQLTablePrivileges** function accepts the following arguments.v2

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle.
UCHAR FAR *	<i>szTableQualifier</i>	Input	Table qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i> .
UCHAR FAR *	<i>szTableOwner</i>	Input	String search pattern for owner names. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i> .
UCHAR FAR *	<i>szTableName</i>	Input	String search pattern for table names.
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR, or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLTablePrivileges** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLTablePrivileges** and explains each one in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.

(1 of 2)

SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned <b>SQL_NEED_DATA</b> . This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name-length arguments was less than 0, but not equal to <b>SQL_NTS</b> .  The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name.
S1C00	Driver not capable	A table qualifier was specified, but the driver or data source does not support qualifiers.  A table owner was specified, but the driver or data source does not support owners.  A string search pattern was specified for the table owner, table name, or column name and the data source does not support search patterns for one or more of those arguments.  The combination of the current settings of the <b>SQL_CONCURRENCY</b> and <b>SQL_CURSOR_TYPE</b> statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the result set. The time-out period is set through <b>SQLSetStmtOption</b> , <b>SQL_QUERY_TIMEOUT</b> .

(2 of 2)

## Usage

The *szTableName* and *szTableOwner* arguments accept search patterns. For more information about valid search patterns, see [“Search Pattern Arguments” on page 13-8](#).



**SQLTablePrivileges** returns the results as a standard result set, ordered by TABLE\_QUALIFIER, TABLE\_OWNER, TABLE\_NAME, and PRIVILEGE. The following table lists the columns in the result set.

**Important:** *SQLTablePrivileges might not return privileges for all tables. Applications can use any valid table, regardless of whether it is returned by SQLTablePrivileges.*

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE\_QUALIFIER, TABLE\_OWNER, and TABLE\_NAME columns, an application can call **SQLGetInfo** with the SQL\_MAX\_QUALIFIER\_NAME\_LEN, SQL\_MAX\_OWNER\_NAME\_LEN, and SQL\_MAX\_TABLE\_NAME\_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	VARCHAR(128)	Table-qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers. .
TABLE_OWNER	VARCHAR(128)	Table-owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	VARCHAR(128) not NULL	Table identifier. .
GRANTOR	VARCHAR(128)	Identifier of the user who granted the privilege; NULL if not applicable to the data source.
GRANTEE	VARCHAR(128) not NULL	Identifier of the user to whom the privilege was granted.

(1 of 2)

Column Name	Data Type	Comments
PRIVILEGE	VARCHAR(128) not NULL	<p>Identifies the table privilege. Can be one of the following or a data-source-specific privilege.</p> <p>SELECT: The grantee is permitted to retrieve data for one or more columns of the table.</p> <p>INSERT: The grantee is permitted to insert new rows containing data for one or more columns into to the table.</p> <p>UPDATE: The grantee is permitted to update the data in one or more columns of the table.</p> <p>DELETE: The grantee is permitted to delete rows of data from the table.</p> <p>REFERENCES: The grantee is permitted to refer to one or more columns of the table within a constraint (for example, a unique, referential, or table-check constraint).</p> <p>The scope of action permitted the grantee by a given table privilege is data source-dependent. For example, the UPDATE privilege might permit the grantee to update all columns in a table on one data source and only those columns for which the grantor has the UPDATE privilege on another data source.</p>
IS_GRANTABLE	VARCHAR(3)	Indicates whether the grantee is permitted to grant the privilege to other users; "YES", "NO", or NULL if unknown or not applicable to the data source.

(2 of 2)

## Code Example

For a code example of a similar function, see [SQLColumns](#).



## Related Functions

---

<b>For information about</b>	<b>See</b>
Assigning storage for a column in a result set	<b>SQLBindCol</b>
Canceling statement processing	<b>SQLCancel</b>
Returning privileges for a column or columns	<b>SQLColumnPrivileges</b>
Returning the columns in a table or tables	<b>SQLColumns</b>
Fetching a block of data or scrolling through a result set	<b>SQLExtendedFetch</b>
Fetching a row of data	<b>SQLFetch</b>
Returning table statistics and indexes	<b>SQLStatistics</b>
Returning a list of tables in a data source	<b>SQLTables</b>

---

## SQLTables

**SQLTables** returns the list of table names that are stored in a specific data source. The driver returns this information as a result set.

### Syntax

```
RETCODE SQLTables(hstmt, szTableQualifier, cbTableQualifier,
szTableOwner, cbTableOwner, szTableName, cbTableName,
szTableType, cbTableType)
```

The **SQLTables** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	<i>hstmt</i>	Input	Statement handle for retrieved results.
UCHAR FAR *	<i>szTableQualifier</i>	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	<i>cbTableQualifier</i>	Input	Length of <i>szTableQualifier</i> .
UCHAR FAR *	<i>szTableOwner</i>	Input	String search pattern for owner names.
SWORD	<i>cbTableOwner</i>	Input	Length of <i>szTableOwner</i> .
UCHAR FAR *	<i>szTableName</i>	Input	String search pattern for table names. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.

(1 of 2)

Type	Argument	Use	Description
SWORD	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .
UCHAR FAR *	<i>szTableType</i>	Input	List of table types to match.
SWORD	<i>cbTableType</i>	Input	Length of <i>szTableType</i> .

(2 of 2)

## Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_STILL\_EXECUTING, SQL\_ERROR or SQL\_INVALID\_HANDLE

## Diagnostics

When **SQLTables** returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, an associated SQLSTATE value can be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLTables** and explains each value in the context of this function; the notation “(DM)” precedes the description of each SQLSTATE returned by the driver manager. The return code associated with each SQLSTATE value is SQL\_ERROR unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication-link failure	The communication link between the driver and the data source failed before the function completed.
24000	Invalid cursor state	A cursor was already opened on the statement handle.
S1000	General error	An error occurred for which no specific SQLSTATE existed and for which no implementation-specific SQLSTATE was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.

(1 of 2)

SQLSTATE	Error	Description
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.
S1008	Operation canceled	The function was called, but before it completed execution, <b>SQLCancel</b> was called on the <i>hstmt</i> from a different thread in a multi-threaded application.
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for the <i>hstmt</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name-length arguments was less than 0, but not equal to SQL_NTS.  The value of one of the name-length arguments exceeded the maximum-length value for the corresponding qualifier or name.
S1C00	Driver not capable	A table qualifier was specified, but the driver or data source does not support qualifiers.  A table owner was specified, but the driver or data source does not support owners.  A string search pattern was specified for the table owner or table name, but the data source does not support search patterns for one or more of those arguments.  The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Time-out expired	The time-out period expired before the data source returned the requested result set. The time-out period is set through <b>SQLSetStmtOption</b> , SQL_QUERY_TIMEOUT.

(2 of 2)

## Usage

**SQLTables** lists all the tables in the requested range. A user might or might not have SELECT privileges to any of these tables. To check accessibility, an application can perform one of the following actions:

- Call **SQLGetInfo** and check the SQL\_ACCESSIBLE\_TABLES info value.
- Call **SQLTablePrivileges** to check the privileges for each table.

Otherwise, the application must handle situations where the user selects a table for which SELECT privileges are not granted.

The *szTableOwner* and *szTableName* arguments accept search patterns. For more information about valid search patterns, see [“Search Pattern Arguments” on page 13-8](#).

To support enumeration of qualifiers, owners, and table types, **SQLTables** defines the following special semantics for the *szTableQualifier*, *szTableOwner*, *szTableName*, and *szTableType* arguments:

- If *szTableQualifier* is a single percent character (%) and *szTableOwner* and *szTableName* are empty strings, the result set contains a list of valid qualifiers for the data source. (All the columns except the TABLE\_QUALIFIER column contain NULLs.)
- If *szTableOwner* is a single percent character (%) and *szTableQualifier* and *szTableName* are empty strings, the result set contains a list of valid owners for the data source. (All the columns except the TABLE\_OWNER column contain NULLs.)
- If *szTableType* is a single percent character (%), and *szTableQualifier*, *szTableOwner*, and *szTableName* are empty strings, then the result set contains a list of valid table types for the data source. (All the columns except the TABLE\_TYPE column contain NULLs.)

If *szTableType* is not an empty string, it must contain a list of comma-separated values for the types of interest; each value must be enclosed in single quotes (') or unquoted. For example, “TABLE', 'VIEW” or “TABLE, VIEW.” If the data source does not support a specified table type, **SQLTables** does not return any results for that type.



**SQLTables** returns the results as a standard result set, ordered by TABLE\_TYPE, TABLE\_QUALIFIER, TABLE\_OWNER, and TABLE\_NAME. The following table lists the columns in the result set.

**Important:** *SQLTables* might not return all qualifiers, owners, or tables. Applications can use any valid qualifier, owner, or table, regardless of whether it is returned by *SQLTables*.

The lengths of VARCHAR columns shown in the following table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE\_QUALIFIER, TABLE\_OWNER, and TABLE\_NAME columns, an application can call **SQLGetInfo** with the SQL\_MAX\_QUALIFIER\_NAME\_LEN, SQL\_MAX\_OWNER\_NAME\_LEN, and SQL\_MAX\_TABLE\_NAME\_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	VARCHAR(128)	Table-qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	VARCHAR(128)	Table-owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	VARCHAR(128)	Table identifier.
TABLE_TYPE	VARCHAR(128)	Table-type identifier; one of the following: "TABLE," "VIEW," "SYSTEM TABLE," "GLOBAL TEMPORARY," "LOCAL TEMPORARY," "ALIAS," "SYNONYM," or a data-source-specific type identifier.
REMARKS	VARCHAR(254)	A description of the table.

## Code Example

For a code example of a similar function, see [SQLColumns](#).

## Related Functions

---

For information about	See
Assigning storage for a column in a result set	<a href="#">SQLBindCol</a>
Canceling statement processing	<a href="#">SQLCancel</a>
Returning privileges for a column or columns	<a href="#">SQLColumnPrivileges</a>
Returning the columns in a table or tables	<a href="#">SQLColumns</a>
Fetching a block of data or scrolling through a result set	<a href="#">SQLExtendedFetch</a>
Fetching a row of data	<a href="#">SQLFetch</a>
Returning table statistics and indexes	<a href="#">SQLStatistics</a>
Returning privileges for a table or tables	<a href="#">SQLTablePrivileges</a>

---

## SQLTransact

**SQLTransact** requests a commit or rollback operation for all active operations on all *hstmts* associated with a connection. **SQLTransact** can also request that a commit or rollback operation be performed for all connections associated with the *henv*.

### Syntax

```
RETCODE SQLTransact(henv, hdbc, fType)
```

The **SQLTransact** function accepts the following arguments.

Type	Argument	Use	Description
HENV	<i>henv</i>	Input	Environment handle
HDBC	<i>hdbc</i>	Input	Connection handle
UWORD	<i>fType</i>	Input	One of the following two values:  SQL_COMMIT SQL_ROLLBACK

### Return Codes

SQL\_SUCCESS, SQL\_SUCCESS\_WITH\_INFO, SQL\_ERROR, or SQL\_INVALID\_HANDLE



## Diagnostics

When **SQLTransact** returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, an associated `SQLSTATE` value can be obtained by calling **SQLError**. The following table lists the `SQLSTATE` values commonly returned by **SQLTransact** and explains each value in the context of this function; the notation “(DM)” precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	INFORMIX-CLI informational message (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection not open	(DM) The <i>hdbc</i> was not in a connected state.
08007	Connection failure during transaction	The connection associated with the <i>hdbc</i> failed during the execution of the function, and it cannot be determined whether the requested COMMIT or ROLLBACK occurred before the failure.
S1000	General error	An error occurred for which no specific <code>SQLSTATE</code> existed and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <b>SQLError</b> in the argument <i>szErrorMsg</i> describes the error and its cause.
S1001	Memory-allocation failure	The driver did not allocate memory required to support execution or completion of the function.

(1 of 2)

SQLSTATE	Error	Description
S1010	Function-sequence error	(DM) <b>SQLExecute</b> or <b>SQLExecDirect</b> was called for an <i>hstmt</i> associated with the <i>hdbc</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1012	Invalid transaction operation code	(DM) The value specified for the argument <i>fType</i> was neither SQL_COMMIT nor SQL_ROLLBACK.
S1C00	Driver not capable	The driver or data source does not support the ROLLBACK operation.

(2 of 2)

## Usage

If *hdbc* is SQL\_NULL\_HDBC and *henv* is a valid environment handle, the driver manager attempts to commit or roll back transactions on all *hdbcs* that are in a connected state. The driver manager calls **SQLTransact** in the driver associated with each *hdbc*. The driver manager returns SQL\_SUCCESS only if it receives SQL\_SUCCESS for each *hdbc*. If the driver manager receives SQL\_ERROR on one or more *hdbcs*, it returns SQL\_ERROR to the application. To determine which connection(s) failed during the COMMIT or ROLLBACK operation, the application can call **SQLError** for each *hdbc*.



**Important:** *The driver manager does not simulate a global transaction across all hdbcs and therefore does not use two-phase commit protocols.*

If *hdbc* is a valid connection handle, *henv* is ignored, and the driver manager calls **SQLTransact** in the driver for the *hdbc*.

If *hdbc* is SQL\_NULL\_HDBC and *henv* is SQL\_NULL\_HENV, **SQLTransact** returns SQL\_INVALID\_HANDLE.

If *fType* is SQL\_COMMIT, **SQLTransact** issues a COMMIT request for all active operations on any *hstmt* associated with an affected *hdbc*.

If *fType* is SQL\_ROLLBACK, **SQLTransact** issues a ROLLBACK request for all active operations on any *hstmt* associated with an affected *hdbc*. If no transactions are active, **SQLTransact** returns SQL\_SUCCESS with no effect on any data sources.

If the driver is in manual-commit mode (by calling **SQLSetConnectOption** with the `SQL_AUTOCOMMIT` option set to zero), a new transaction is implicitly started when an SQL statement that can be contained within a transaction is executed against the current data source.

To determine how transaction operations affect cursors, an application calls **SQLGetInfo** with the `SQL_CURSOR_ROLLBACK_BEHAVIOR` and `SQL_CURSOR_COMMIT_BEHAVIOR` options.

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

If the value of `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` equals `SQL_CB_CLOSE`, **SQLTransact** closes all open cursors on all *hstmts* associated with the *hdbc*. **SQLTransact** leaves any *hstmt* present in a prepared state; the application can call **SQLExecute** for an *hstmt* associated with the *hdbc* without first calling **SQLPrepare**.

If the value of `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` equals `SQL_CB_PRESERVE`, **SQLTransact** does not affect open cursors associated with the *hdbc*. Cursors remain at the row to which they pointed before the call to **SQLTransact**.

For drivers and data sources that support transactions, calling **SQLTransact** with either `SQL_COMMIT` or `SQL_ROLLBACK` when no transaction is active returns `SQL_SUCCESS` (indicating that there is no work to be committed or rolled back) and has no effect on the data source.

Drivers or data sources that do not support transactions (**SQLGetInfo** *fOption* `SQL_TXN_CAPABLE` is 0) are effectively always in autocommit mode. Therefore, calling **SQLTransact** with `SQL_COMMIT` returns `SQL_SUCCESS`. However, calling **SQLTransact** with `SQL_ROLLBACK` results in `SQLSTATE S1C00` (Driver not capable), indicating that a rollback can never be performed.

## Related Functions

---

<b>For information about</b>	<b>See</b>
Returning information about a driver or data source	<b>SQLGetInfo</b>
Freeing a statement handle	<b>SQLFreeStmt</b>

---

---

# Setup Shared Library Function Reference

ConfigDSN . . . . .	14-3
ConfigTranslator. . . . .	14-7



**T**his chapter describes the syntax of the driver-setup shared library API, which consists of a single function (**ConfigDSN**). **ConfigDSN** can be in the driver shared library or in a separate-setup shared library.

This chapter also describes the syntax of the translator-setup shared library API, which consists of a single function (**ConfigTranslator**). **ConfigTranslator** can be in the translator-setup shared library or in a separate setup shared library.

For information on argument naming conventions, see [“Arguments” on page 13-4](#).

---

## ConfigDSN

**ConfigDSN** adds, modifies, or deletes data sources from the **odbc.ini** file. It might prompt the user for connection information. It can be in the driver shared library or a separate-setup shared library. **ConfigDSN** was introduced in ODBC 1.0

### Syntax

```
BOOL ConfigDSN(hwndParent, fRequest, lpszDriver,  
lpszAttributes)
```

The **ConfigDSN** function accepts the following arguments.

Type	Argument	Use	Description
HWND	<i>hwndParent</i>	Input	Parent window handle. The function will not display any dialog boxes if the handle is null.
UINT	<i>fRequest</i>	Input	Type of request. <i>fRequest</i> must contain one of the following values: <ul style="list-style-type: none"> <li>■ ODBC_ADD_DSN Add a new data source.</li> <li>■ ODBC_CONFIG_DSN Configure (modify) an existing data source.</li> <li>■ ODBC_REMOVE_DSN Remove an existing data source.</li> </ul>
LPCSTR	<i>lpszDriver</i>	Input	Driver description (usually the name of the associated DBMS) presented to users instead of the physical driver name.
LPCSTR	<i>lpszAttributes</i>	Input	List of attributes in the form of keyword-value pairs. For information about the list structure, see “Comments.”

## Returns

The function returns TRUE if it is successful. It returns FALSE if it fails.

## Usage

**ConfigDSN** receives connection information from the installer shared library as a list of attributes in the form of keyword-value pairs. Each pair is terminated with a null byte, and the entire list is null terminated (that is, two null bytes mark the end of the list). The keywords used by **ConfigDSN** are the same as those used by **SQLBrowseConnect** and **SQLDriverConnect**, except **ConfigDSN** does not accept the DRIVER keyword. As in **SQLBrowseConnect** and **SQLDriverConnect**, the keywords and their values should not contain the [ ] { } ( ) , ; ? \* = ! @ \ characters, and the value of the DSN keyword cannot consist of blanks only. Because of the registry grammar, keywords and data-source names cannot contain the backslash (\) character.



For example, to configure a data source that requires a user ID, password, and database name, a setup application might pass the following keyword-value pairs:

```
DSN=Personnel Data\OUID=Smith\OPWD=Sesame\ODATABASE=Personnel\O\O
```

For more information about these keywords, see [“SQLDriverConnect” on page 13-93](#) and [“Adding and Modifying Data Sources” on page 2-6](#).

In order to display a dialog box, *hwndParent* must not be null.

### ***Adding a Data Source***

If a data-source name is passed to **ConfigDSN** in *lpszAttributes*, **ConfigDSN** checks that the name is valid. If the data-source name matches an existing data-source name and *hwndParent* is null, **ConfigDSN** overwrites the existing name. If it matches an existing name and *hwndParent* is not null, **ConfigDSN** prompts the user to overwrite the existing name.

If *lpszAttributes* contains enough information to connect to a data source, **ConfigDSN** can add the data source or display a dialog box with which the user can change the connection information. If *lpszAttributes* does not contain enough information to connect to a data source, **ConfigDSN** must determine the necessary information; if *hwndParent* is not null, it displays a dialog box to retrieve the information from the user.

If **ConfigDSN** displays a dialog box, it must display any connection information passed to it in *lpszAttributes*. In particular, if a data-source name was passed to it, **ConfigDSN** displays that name but does not allow the user to change it. **ConfigDSN** can supply default values for connection information not passed to it in *lpszAttributes*.

If **ConfigDSN** cannot get complete connection information for a data source, it returns FALSE.

If **ConfigDSN** can get complete connection information for a data source, it calls **SQLWriteDSNToIni** in the installer shared library to add the new data-source specification to the **odbc.ini** file. **SQLWriteDSNToIni** adds the data-source name to the ODBC Data Sources section, creates the data-source-specification section, and adds the **Driver** keyword with the driver description as its value. **ConfigDSN** calls **SQLWritePrivateProfileString** in the installer shared library to add more keywords and values that the driver needs.

### ***Modifying a Data Source***

To modify a data source, a data-source name must be passed to **ConfigDSN** in *lpszAttributes*. **ConfigDSN** checks that the data-source name is in the **odbc.ini** file.

If *hwndParent* is null, **ConfigDSN** uses the information in *lpszAttributes* to modify the information in the **odbc.ini** file. If *hwndParent* is not null, **ConfigDSN** displays a dialog box that uses the information in *lpszAttributes*; for information that is not in *lpszAttributes*, it uses information from the **odbc.ini** file. The user can modify the information before **ConfigDSN** stores it in the **odbc.ini** file.

If the data-source name was changed, **ConfigDSN** first calls **SQLRemoveDSNFromIni** in the installer shared library to remove the existing data-source specification from the **odbc.ini** file. It then follows the steps in the previous section to add the new data-source specification. If the data-source name was not changed, **ConfigDSN** calls **SQLWritePrivateProfileString** in the installer shared library to make any other changes. **ConfigDSN** cannot delete or change the value of the **Driver** keyword.

### ***Deleting a Data Source***

To delete a data source, a data-source name must be passed to **ConfigDSN** in *lpszAttributes*. **ConfigDSN** checks that the data-source name is in the **odbc.ini** file. It then calls **SQLRemoveDSNFromIni** in the installer shared library to remove the data source.

## ConfigTranslator

**ConfigTranslator** returns a default translation option for a translator. It can be in the translator shared library or a separate setup shared library.

**ConfigTranslator** was introduced in ODBC 2.0.

### Syntax

```
B00L ConfigTranslator(hwndParent, pvOption)
```

The **ConfigTranslator** function accepts the following arguments.

Type	Argument	Use	Description
HWND	<i>hwndParent</i>	Input	Parent window handle. The function will not display any dialog boxes if the handle is null.
DWORD FAR *	<i>pvOption</i>	Output	A 32-bit translation option.

### Returns

The function returns TRUE if it is successful. It returns FALSE if it fails.

### Comments

If the translator supports only a single translation option, **ConfigTranslator** returns TRUE and sets *pvOption* to the 32-bit option. Otherwise, it determines the default translation option to use. **ConfigTranslator** can display a dialog box with which a user selects a default translation option.

## Related Functions

---

<b>For information about</b>	<b>See</b>
Getting a translation option	<b>SQLGetConnectOption</b>
Selecting a translator	<b>SQLGetTranslator</b>
Setting a translation option	<b>SQLSetConnectOption</b>

---

---

# Translation Shared Library Function Reference

SQLDataSourceToDriver . . . . .	15-3
SQLDriverToDataSource . . . . .	15-7



# T

his chapter describes the syntax of the translation shared library API, which comprises **SQLDriverToDataSource** and **SQLDataSourceToDriver** functions. These functions must be included in the shared library that performs translations for the driver.

For information on argument naming conventions, see [Chapter 13, “INFORMIX-CLI Function Reference.”](#)

---

## SQLDataSourceToDriver

**SQLDataSourceToDriver** supports translations for the Informix driver. This function is not called by INFORMIX-CLI-enabled applications; applications request translation through **SQLSetConnectOption**. The driver associated with the *hdbc* specified in **SQLSetConnectOption** calls the specified shared library to perform translations of all data that flows from the data source to the driver. A default translation shared library can be specified in the **odbc.ini** file.

### Syntax

```
BOOL SQLDataSourceToDriver(fOption, fSqlType, rgbValueIn,
cbValueIn, rgbValueOut, cbValueOutMax, pcbValueOut,
szErrorMsg, cbErrorMsgMax, pcbErrorMsg)
```

The **SQLDataSourceToDriver** function accepts the following arguments.

Type	Argument	Use	Description
UDWORD	<i>fOption</i>	Input	Option value.
SWORD	<i>fSqlType</i>	Input	The ODBC SQL data type. This argument tells the driver how to convert <i>rgbValueIn</i> to a form acceptable by the application. This must be one of the following values:  SQL_CHAR SQL_DATE SQL_DECIMAL SQL_DOUBLE SQL_INTEGER SQL_LONGVARBINARY SQL_LONGVARCHAR SQL_REAL SQL_SMALLINT SQL_TIMESTAMP SQL_VARCHAR  For information on ODBC SQL data types, see <a href="#">Appendix C</a> . For information on how Informix data types map to ODBC SQL data types, see <a href="#">“Mapping Data Types” on page 1-15</a> .
PTR	<i>rgbValueIn</i>	Input	Value to translate.
SDWORD	<i>cbValueIn</i>	Input	Length of <i>rgbValueIn</i> .
PTR	<i>rgbValueOut</i>	Output	Result of the translation.  The translation shared library does not null-terminate this value.
SDWORD	<i>cbValueOutMax</i>	Input	Length of <i>rgbValueOut</i> .

(1 of 2)



Type	Argument	Use	Description
SDWORD FAR*	<i>pcbValueOut</i>	Output	<p>The total number of bytes (excluding the null-termination byte) available to return in <i>rgbValueOut</i>.</p> <p>For character or binary data, if this is greater than or equal to <i>cbValueOutMax</i>, the data in <i>rgbValueOut</i> is truncated to <i>cbValueOutMax</i> bytes.</p> <p>For all other data types, the value of <i>cbValueOutMax</i> is ignored, and the translation shared library assumes the size of <i>rgbValueOut</i> is the size of the default C data type of the SQL data type specified with <i>fSqlType</i>.</p>
UCHAR FAR *	<i>szErrorMsg</i>	Output	<p>Pointer to storage for an error message. This is an empty string unless the translation failed.</p>
SWORD	<i>cbErrorMsgMax</i>	Input	<p>Length of <i>szErrorMsg</i>.</p>
SWORD FAR *	<i>pcbErrorMsg</i>	Output	<p>Pointer to the total number of bytes (excluding the null termination byte) available to return in <i>szErrorMsg</i>. If this is greater than or equal to <i>cbErrorMsg</i>, the data in <i>szErrorMsg</i> is truncated to <i>cbErrorMsgMax</i> - 1 bytes.</p>

(2 of 2)

## Return Codes

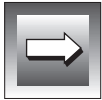
The application returns TRUE if the translation succeeds; FALSE if it fails.

## Usage

The driver calls **SQLDataSourceToDriver** to translate *all* the data (result set data, table names, row counts, error messages, and so on) that passes from the data source to the driver. The translation shared library might not translate some data, depending on the data type and the purpose of the translation shared library. For example, a shared library that translates character data from one code page to another ignores all numeric and binary data.

The value of *fOption* is set to the specified value of *vParam* by calling **SQLSetConnectOption** with the `SQL_TRANSLATE_OPTION` option. It is a 32-bit value that has a specific meaning for a given translation shared library. For example, it might specify a certain character-set translation.

If the same buffer is specified for *rgbValueIn* and *rgbValueOut*, the translation of data in the buffer is performed in place.



**Important:** Although *cbValueIn*, *cbValueOutMax*, and *pcbValueOut* are the `SDWORD` type, **SQLDataSourceToDriver** does not necessarily support huge pointers.

If **SQLDataSourceToDriver** returns `FALSE`, data truncation might have occurred during translation. If *pcbValueOut* (the number of bytes available to return in the output buffer) is greater than *cbValueOutMax* (the length of the output buffer), then truncation occurs. The driver must determine whether the truncation is acceptable. If truncation does not occur, the **SQLDataSourceToDriver** returned `FALSE` due to another error. In either case, a specific error message is returned in *szErrorMsg*.

## Related Functions

For information about	See
Translating data being sent to the data source	<b>SQLDriverToDataSource</b>
Returning the setting of a connection option	<b>SQLGetConnectOption</b> (extension)
Setting a connection option	<b>SQLSetConnectOption</b> (extension)

## SQLDriverToDataSource

**SQLDriverToDataSource** supports translations for the Informix driver. This function is not called by INFORMIX-CLI-enabled applications; applications request translation through **SQLSetConnectOption**. The driver associated with the *hdbc* specified in **SQLSetConnectOption** calls the specified shared library to perform translations of all the data that flows from the driver to the data source. A default translation shared library can be specified in the **odbc.ini** file.

### Syntax

```

BOOL SQLDriverToDataSource(fOption, fSqlType, rgbValueIn,
cbValueIn, rgbValueOut, cbValueOutMax, pcbValueOut,
szErrorMsg, cbErrorMsg, pcbErrorMsg)

```

Level 2

The **SQLDriverToDataSource** function accepts the following arguments.

Type	Argument	Use	Description
UDWORD	<i>fOption</i>	Input	Option value.
SDWORD	<i>fSqlType</i>	Input	<p>The ODBC SQL data type. This argument tells the driver how to convert <i>rgbValueIn</i> to a form acceptable by the data source. This must be one of the following values:</p> <p>SQL_CHAR            SQL_DATE            SQL_DECIMAL            SQL_DOUBLE            SQL_INTEGER            SQL_LONGVARBINARY            SQL_LONGVARCHAR            SQL_REAL            SQL_SMALLINT            SQL_TIMESTAMP            SQL_VARCHAR</p> <p>For information on ODBC SQL data types, see <a href="#">Appendix C</a>. For information on how Informix data types map to ODBC SQL data types, see “<a href="#">Mapping Data Types</a>” on page 1-15.</p>
PTR	<i>rgbValueIn</i>	Input	Value to translate.
SDWORD	<i>cbValueIn</i>	Input	Length of <i>rgbValueIn</i> .
PTR	<i>rgbValueOut</i>	Output	Result of the translation. The translation shared library does not null-terminate this value.
SDWORD	<i>cbValueOutMax</i>	Input	Length of <i>rgbValueOut</i> .

(1 of 2)

Type	Argument	Use	Description
SDWORD FAR *	<i>pcbValueOut</i>	Output	<p>The total number of bytes (excluding the null-termination byte) available to return in <i>rgbValueOut</i>.</p> <p>For character or binary data, if this is greater than or equal to <i>cbValueOutMax</i>, the data in <i>rgbValueOut</i> is truncated to <i>cbValueOutMax</i> bytes.</p> <p>For all other data types, the value of <i>cbValueOutMax</i> is ignored and the translation shared library assumes the size of <i>rgbValueOut</i> is the size of the default C data type of the SQL data type specified with <i>iSqlType</i>.</p>
UCHAR FAR *	<i>szErrorMsg</i>	Output	Pointer to storage for an error message. This is an empty string unless the translation failed.
SWORD	<i>cbErrorMsgMax</i>	Input	Length of <i>szErrorMsg</i> .
SWORD FAR *	<i>pcbErrorMsg</i>	Output	Pointer to the total number of bytes (excluding the null termination byte) available to return in <i>szErrorMsg</i> . If this is greater than or equal to <i>cbErrorMsg</i> , the data in <i>szErrorMsg</i> is truncated to <i>cbErrorMsgMax</i> - 1 bytes.

(2 of 2)

## Return Codes

The application returns TRUE if the translation succeeds; FALSE if it fails.

## Usage

The driver calls **SQLDriverToDataSource** to translate all data (SQL statements, parameters, and so on) that passes from the driver to the data source. The translation shared library might not translate some data, depending on the data type and the purpose of the translation shared library. For example, a shared library that translates character data from one code page to another ignores all numeric and binary data.

The value of *fOption* is set to the value of *vParam* that is specified by calling **SQLSetConnectOption** with the `SQL_TRANSLATE_OPTION` option. It is a 32-bit value that has a specific meaning for a given translation shared library. For example, it could specify a certain character-set translation.

If the same buffer is specified for *rgbValueIn* and *rgbValueOut*, the translation of data in the buffer is performed in place.

**Important:** Although *cbValueIn*, *cbValueOutMax*, and *pcbValueOut* are the *SDWORD* type, **SQLDriverToDataSource** does not necessarily support huge pointers.

If **SQLDriverToDataSource** returns `FALSE`, data truncation might have occurred during translation. If *pcbValueOut* (the number of bytes available to return in the output buffer) is greater than *cbValueOutMax* (the length of the output buffer), then truncation occurs. The driver must determine whether the truncation is acceptable. If truncation does not occur, the **SQLDriverToDataSource** returned `FALSE` due to another error. In either case, a specific error message is returned in *szErrorMsg*.

## Related Functions

For information about	See
Translating data returned from the data source	<b>SQLDataSourceToDriver</b>
Returning the setting of a connection option	<b>SQLGetConnectOption</b> )
Setting a connection option	<b>SQLSetConnectOption</b> )



---

# INFORMIX-CLI Error Codes

**SQLError** returns `SQLSTATE` values as defined by the X/Open and SQL Access Group SQL CAE specification (1992). `SQLSTATE` values are strings that contain five characters. The following table lists `SQLSTATE` values that the Informix driver can return for **SQLError**.

The character-string value returned for an `SQLSTATE` consists of a two-character class value followed by a three-character subclass value. A class value of "01" indicates a warning and is accompanied by a return code of `SQL_SUCCESS_WITH_INFO`. Class values other than "01," except for the class "IM," indicate an error and are accompanied by a return code of `SQL_ERROR`. The class "IM" is specific to warnings and errors that derive from the implementation of INFORMIX-CLI. The subclass value "000" in any class is for implementation-defined conditions within the given class. The assignment of class and subclass values is defined by ANSI SQL-92.

A return value of `SQL_SUCCESS` normally indicates a function has executed successfully, although the `SQLSTATE 00000` also indicates success.

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
01000	General warning	All INFORMIX-CLI functions except:  <b>SQLAllocEnv</b> <b>SQLError</b>
01002	Disconnect error	<b>SQLDisconnect</b>
01004	Data truncated	<b>SQLBrowseConnect</b> <b>SQLColAttributes</b> <b>SQLDataSources</b> <b>SQLDescribeCol</b> <b>SQLDriverConnect</b> <b>SQLDrivers</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetCursorName</b> <b>SQLGetData</b> <b>SQLGetInfo</b> <b>SQLNativeSql</b> <b>SQLPutData</b>
01006	Privilege not revoked	<b>SQLExecDirect</b> <b>SQLExecute</b>
01S00	Invalid connection string attribute	<b>SQLBrowseConnect</b> <b>SQLDriverConnect</b>
01S01	Error in row	<b>SQLExtendedFetch</b>
01S02	Option value changed	<b>SQLSetConnectOption</b> <b>SQLSetStmtOption</b>
01S03	No rows updated or deleted	<b>SQLExecDirect</b> <b>SQLExecute</b>

(1 of 13)



<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
01S04	More than one row updated or deleted	<b>SQLExecDirect</b> <b>SQLExecute</b>
07001	Wrong number of parameters	<b>SQLExecDirect</b> <b>SQLExecute</b>
07006	Restricted data-type-attribute violation	<b>SQLBindParameter</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b>
08001	Unable to connect to data source	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
08002	Connection in use	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b> <b>SQLSetConnectOption</b>
08003	Connection not open	<b>SQLAllocStmt</b> <b>SQLDisconnect</b> <b>SQLGetConnectOption</b> <b>SQLGetInfo</b> <b>SQLNativeSql</b> <b>SQLSetConnectOption</b> <b>SQLTransact</b>
08004	Data source rejected establishment of connection	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
08007	Connection failure during transaction	<b>SQLTransact</b>

(2 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
08S01	Communication-link failure	<b>SQLBrowseConnect</b> <b>SQLColumnPrivileges</b> <b>SQLColumns</b> <b>SQLConnect</b> <b>SQLDriverConnect</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLFreeConnect</b> <b>SQLGetData</b> <b>SQLGetTypeInfo</b> <b>SQLParamData</b> <b>SQLPrepare</b> <b>SQLPrimaryKeys</b> <b>SQLProcedures</b> <b>SQLPutData</b> <b>SQLSetConnectOption</b> <b>SQLSetStmtOption</b> <b>SQLSpecialColumns</b> <b>SQLStatistics</b> <b>SQLTablePrivileges</b> <b>SQLTables</b>
21S01	Insert value list does not match column list	<b>SQLExecDirect</b> <b>SQLPrepare</b>
21S02	Degree of derived table does not match column list	<b>SQLExecDirect</b> <b>SQLPrepare</b>
22001	String data right truncation	<b>SQLPutData</b>
22002	Indicator variable required but not supplied	<b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b>

(3 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
22003	Numeric value out of range	<b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b> <b>SQLGetInfo</b> <b>SQLPutData</b>
22005	Error in assignment	<b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b> <b>SQLPrepare</b> <b>SQLPutData</b>
22008	DATETIME field overflow	<b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b> <b>SQLPutData</b>
22012	Division by zero	<b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b>
22026	String data, length mismatch	<b>SQLParamData</b>
23000	Integrity-constraint violation	<b>SQLExecDirect</b> <b>SQLExecute</b>

(4 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
24000	Invalid cursor state	<b>SQLColAttributes</b> <b>SQLColumnPrivileges</b> <b>SQLColumns</b> <b>SQLDescribeCol</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b> <b>SQLGetStmtOption</b> <b>SQLGetTypeInfo</b> <b>SQLPrepare</b> <b>SQLPrimaryKeys</b> <b>SQLProcedures</b> <b>SQLSetCursorName</b> <b>SQLSetStmtOption</b> <b>SQLSpecialColumns</b> <b>SQLStatistics</b> <b>SQLTablePrivileges</b> <b>SQLTables</b>
25000	Invalid transaction state	<b>SQLDisconnect</b>
28000	Invalid authorization specification	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
34000	Invalid cursor name	<b>SQLExecDirect</b> <b>SQLPrepare</b> <b>SQLSetCursorName</b>
37000	Syntax error or access violation	<b>SQLExecDirect</b> <b>SQLNativeSql</b> <b>SQLPrepare</b>
3C000	Duplicate cursor name	<b>SQLSetCursorName</b>

(5 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
40001	Serialization failure	<b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b>
42000	Syntax error or access violation	<b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLPrepare</b>
70100	Operation aborted	<b>SQLCancel</b>
IM002	Data-source name not found and no default driver specified	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
IM003	Specified driver could not be loaded	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
IM004	Driver <b>SQLAllocEnv</b> failed	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
IM005	Driver <b>SQLAllocConnect</b> failed	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
IM006	Driver <b>SQLSetConnectOption</b> failed	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b>
IM007	No data source or driver specified; dialog prohibited	<b>SQLDriverConnect</b>
IM008	Dialog failed	<b>SQLDriverConnect</b>
IM009	Unable to load translation shared library	<b>SQLBrowseConnect</b> <b>SQLConnect</b> <b>SQLDriverConnect</b> <b>SQLSetConnectOption</b>

(6 of 13)

<b>SQLSTATEs</b>	<b>Error</b>	<b>Can be returned from</b>
IM010	Data-source name too long	<b>SQLBrowseConnect</b> <b>SQLDriverConnect</b>
IM011	Driver name too long	<b>SQLBrowseConnect</b> <b>SQLDriverConnect</b>
IM012	DRIVER keyword syntax error	<b>SQLBrowseConnect</b> <b>SQLDriverConnect</b>
IM013	Trace file error	All ODBC functions.
S0001	Base table or view already exists	<b>SQLExecDirect</b> <b>SQLPrepare</b>
S0002	Base table not found	<b>SQLExecDirect</b> <b>SQLPrepare</b>
S0011	Index already exists	<b>SQLExecDirect</b> <b>SQLPrepare</b>
S0012	Index not found	<b>SQLExecDirect</b> <b>SQLPrepare</b>
S0021	Column already exists	<b>SQLExecDirect</b> <b>SQLPrepare</b>
S0022	Column not found	<b>SQLExecDirect</b> <b>SQLPrepare</b>
S1000	General error	All ODBC functions except: <b>SQLAllocEnv</b> <b>SQLError</b>
S1001	Memory-allocation failure	All ODBC functions except:  <b>SQLAllocEnv</b> <b>SQLError</b> <b>SQLFreeConnect</b> <b>SQLFreeEnv</b>

(7 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
S1002	Invalid column number	<b>SQLBindCol</b> <b>SQLColAttributes</b> <b>SQLDescribeCol</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b>
S1003	Program type out of range	<b>SQLBindCol</b> <b>SQLBindParameter</b> <b>SQLGetData</b>
S1004	SQL data type out of range	<b>SQLBindParameter</b> <b>SQLGetTypeInfo</b>
S1009	Invalid argument value	<b>SQLAllocConnect</b> <b>SQLAllocStmt</b> <b>SQLBindCol</b> <b>SQLBindParameter</b> <b>SQLExecDirect</b> <b>SQLGetData</b> <b>SQLGetInfo</b> <b>SQLNativeSql</b> <b>SQLPrepare</b> <b>SQLPutData</b> <b>SQLSetConnectOption</b> <b>SQLSetCursorName</b> <b>SQLSetStmtOption</b>

(8 of 13)

<b>SQLSTATEs</b>	<b>Error</b>	<b>Can be returned from</b>
S1010	Function-sequence error	<b>SQLBindCol</b> <b>SQLBindParameter</b> <b>SQLColAttributes</b> <b>SQLColumnPrivileges</b> <b>SQLColumns</b> <b>SQLDescribeCol</b> <b>SQLDisconnect</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLFreeConnect</b> <b>SQLFreeEnv</b> <b>SQLFreeStmt</b> <b>SQLGetConnectOption</b> <b>SQLGetCursorName</b> <b>SQLGetData</b> <b>SQLGetFunctions</b> <b>SQLGetStmtOption</b> <b>SQLGetTypeInfo</b> <b>SQLMoreResults</b> <b>SQLNumParams</b> <b>SQLNumResultCols</b> <b>SQLParamData</b> <b>SQLParamOptions</b> <b>SQLPrepare</b> <b>SQLPrimaryKeys</b> <b>SQLProcedures</b> <b>SQLPutData</b> <b>SQLRowCount</b> <b>SQLSetConnectOption</b> <b>SQLSetCursorName</b> <b>SQLSetScrollOptions</b> <b>SQLSetStmtOption</b> <b>SQLSpecialColumns</b> <b>SQLStatistics</b> <b>SQLTablePrivileges</b> <b>SQLTables</b> <b>SQLTransact</b>
S1011	Operation invalid at this time	<b>SQLGetStmtOption</b> <b>SQLSetConnectOption</b> <b>SQLSetStmtOption</b>



<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
S1012	Invalid transaction operation code specified	<b>SQLTransact</b>
S1015	No cursor name available	<b>SQLGetCursorName</b>
S1090	Invalid string or buffer length	<b>SQLBindCol</b> <b>SQLBindParameter</b> <b>SQLBrowseConnect</b> <b>SQLColAttributes</b> <b>SQLColumnPrivileges</b> <b>SQLColumns</b> <b>SQLConnect</b> <b>SQLDataSources</b> <b>SQLDescribeCol</b> <b>SQLDriverConnect</b> <b>SQLDrivers</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLGetCursorName</b> <b>SQLGetData</b> <b>SQLGetInfo</b> <b>SQLNativeSql</b> <b>SQLPrepare</b> <b>SQLPrimaryKeys</b> <b>SQLProcedures</b> <b>SQLPutData</b> <b>SQLSetCursorName</b> <b>SQLSpecialColumns</b> <b>SQLStatistics</b> <b>SQLTablePrivileges</b> <b>SQLTables</b>
S1091	Descriptor type out of range	<b>SQLColAttributes</b>
S1092	Option type out of range	<b>SQLFreeStmt</b> <b>SQLGetConnectOption</b> <b>SQLGetStmtOption</b> <b>SQLSetConnectOption</b> <b>SQLSetStmtOption</b>
S1093	Invalid parameter number	<b>SQLBindParameter</b>
S1094	Invalid scale value	<b>SQLBindParameter</b>

(10 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
S1095	Function type out of range	<b>SQLGetFunctions</b>
S1096	Information type out of range	<b>SQLGetInfo</b>
S1097	Column type out of range	<b>SQLSpecialColumns</b>
S1098	Scope type out of range	<b>SQLSpecialColumns</b>
S1099	Nullable type out of range	<b>SQLSpecialColumns</b>
S1100	Uniqueness option type out of range	<b>SQLStatistics</b>
S1101	Accuracy option type out of range	<b>SQLStatistics</b>
S1103	Direction option out of range	<b>SQLDataSources</b> <b>SQLDrivers</b>
S1104	Invalid precision value	<b>SQLBindParameter</b>
S1105	Invalid parameter type	<b>SQLBindParameter</b>
S1106	Fetch type out of range	<b>SQLExtendedFetch</b>
S1107	Row value out of range	<b>SQLExtendedFetch</b> <b>SQLParamOptions</b> <b>SQLSetScrollOptions</b>
S1108	Concurrency option out of range	<b>SQLSetScrollOptions</b>
S1109	Invalid cursor position	<b>SQLExecute</b> <b>SQLExecDirect</b> <b>SQLGetData</b> <b>SQLGetStmtOption</b>
S1110	Invalid driver completion	<b>SQLDriverConnect</b>

(11 of 13)

<b>SQLSTATES</b>	<b>Error</b>	<b>Can be returned from</b>
S1111	Invalid bookmark value	<b>SQLExtendedFetch</b>
S1C00	Driver not capable	<b>SQLBindCol</b> <b>SQLBindParameter</b> <b>SQLColAttributes</b> <b>SQLColumnPrivileges</b> <b>SQLColumns</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetConnectOption</b> <b>SQLGetData</b> <b>SQLGetInfo</b> <b>SQLGetStmtOption</b> <b>SQLGetTypeInfo</b> <b>SQLPrepare</b> <b>SQLPrimaryKeys</b> <b>SQLProcedures</b> <b>SQLSetConnectOption</b> <b>SQLSetScrollOptions</b> <b>SQLSetStmtOption</b> <b>SQLSpecialColumns</b> <b>SQLStatistics</b> <b>SQLTablePrivileges</b> <b>SQLTables</b> <b>SQLTransact</b>

(12 of 13)

SQLSTATES	Error	Can be returned from
S1T00	Time-out expired	<b>SQLBrowseConnect</b> <b>SQLColAttributes</b> <b>SQLColumnPrivileges</b> <b>SQLColumns</b> <b>SQLConnect</b> <b>SQLDescribeCol</b> <b>SQLDriverConnect</b> <b>SQLExecDirect</b> <b>SQLExecute</b> <b>SQLExtendedFetch</b> <b>SQLFetch</b> <b>SQLGetData</b> <b>SQLGetInfo</b> <b>SQLGetTypeInfo</b> <b>SQLMoreResults</b> <b>SQLNumParams</b> <b>SQLNumResultCols</b> <b>SQLParamData</b> <b>SQLPrepare</b> <b>SQLPrimaryKeys</b> <b>SQLProcedures</b> <b>SQLPutData</b> <b>SQLSpecialColumns</b> <b>SQLStatistics</b> <b>SQLTablePrivileges</b> <b>SQLTables</b>

(13 of 13)

# SQL Grammar

This chapter describes the recommended syntax for maximum interoperability in calls to **SQLPrepare**, **SQLExecute**, and **SQLExecDirect**. To the right of each construct is an indicator that tells whether the construct is part of the minimum grammar, the core grammar, or the extended grammar



**Important:** *Informix database servers provide extensions to SQL other than those defined by ODBC. To use one of these extensions, refer to [Informix Guide to SQL: Syntax](#). If your application uses the Informix-specific syntax, it is not interoperable among DBMSs.*

The Integrity Enhancement Facility (IEF) is included in the grammar but is optional. The grammar for the IEF comes from the X/Open and SQL Access Group SQL CAE specification (1992) and is a subset of the ISO SQL-92 standard. Elements that are part of the IEF and are optional in the ANSI 1989 standard are presented in the following typeface and font, distinct from the rest of the grammar:

```
table-constraint-definition
```



**Important:** *In `CREATE TABLE` and `ALTER TABLE` statements, applications must use the data type name returned by `SQLGetTypeInfo` in the `TYPE_NAME` column.*

---

## Parameter Data Types

Although each parameter specified with **SQLBindParameter** is defined using an SQL data type, the parameters in an SQL statement have no intrinsic data type. Therefore, parameter markers can be included in an SQL statement only if their data types can be inferred from another operand in the statement. For example, in an arithmetic expression such as `? + COLUMN1`, the data type of the parameter can be inferred from the data type of the named column represented by `COLUMN1`. An application cannot use a parameter marker if the data type cannot be determined.

The following table describes how a data type is determined for several types of parameters.

---

Location of Parameter	Assumed Data Type
One operand of a binary arithmetic or comparison operator	Same as the other operand
The first operand in a BETWEEN clause	Same as the other operand
The second or third operand in a BETWEEN clause	Same as the first operand
An expression used with IN	Same as the first value or the result column of the subquery
A value used with IN	Same as the expression
A pattern value used with LIKE	VARCHAR
An update value used with UPDATE	Same as the update column

---

---

## Parameter Markers

An application cannot place parameter markers in the following locations:

- In a SELECT list
- As both *expressions* in a *comparison-predicate*
- As both operands of a binary operator
- As both the first and second operands of a BETWEEN operation
- As both the first and third operands of a BETWEEN operation
- As both the expression and the first value of an IN operation
- As the operand of a unary + or – operation
- As the argument of a *set-function-reference*

For more information, see the ANSI SQL-92 specification.

If an application includes parameter markers in the SQL statement, the application must call **SQLBindParameter** to associate storage locations with parameter markers before it calls **SQLExecute** or **SQLExecDirect**. If the application calls **SQLPrepare**, the application can call **SQLBindParameter** before or after it calls **SQLPrepare**.

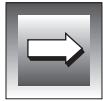
The application can set parameter markers in any order. The driver buffers argument descriptors and sends the current values referenced by the **SQLBindParameter** argument *rgbValue* for the associated parameter marker when the application calls **SQLExecute** or **SQLExecDirect**. It is the responsibility of the application to ensure that all pointer arguments are valid at execution time.

The keyword USER in the following tables represents a character string containing the *user-name* of the current user.

## SQL Statements

The following SQL statements define the base ODBC SQL grammar.

### ALTER TABLE Statement



**Important:** As a data-type in an ALTER TABLE statement, applications must use a data type from the TYPE\_NAME column of the result set that *SQLGetTypeInfo* returns.

**SQL conformance level:** Core

```
ALTER TABLE base-table-name
{ADD column-identifier data-type
| ADD (column-identifier data-type [, column-identifier data-type]...)
}
```

**SQL conformance level:** Extended

```
ALTER TABLE base-table-name
{ADD column-identifier data-type
| ADD (column-identifier data-type [, column-identifier data-type]...)
| DROP [COLUMN] column-identifier [CASCADE | RESTRICT]
}
```

### CREATE INDEX Statement

**SQL conformance level:** Core

```
CREATE [UNIQUE] INDEX index-name
ON base-table-name
( column-identifier [ASC | DESC]
[, column-identifier [ASC | DESC] ]... )
```





## CREATE TABLE Statement

**Important:** As a data-type in a CREATE TABLE statement, applications must use a data type from the `TYPE_NAME` column of the result that `SQLGetTypeInfo` returns.

SQL conformance level: Minimum

CREATE TABLE *base-table-name*  
(*column-element* [, *column-element*] ...)

*column-element* ::= *column-definition* | *table-constraint-definition*

*column-definition* ::=

*column-identifier* *data-type*

[DEFAULT *default-value*]

[*column-constraint-definition* [*column-constraint-definition*]...]

*default-value* ::= *literal* | NULL | USER

*column-constraint-definition* ::=

NOT NULL

| UNIQUE | PRIMARY KEY

| REFERENCES *ref-table-name* *referenced-columns*

| CHECK (*search-condition*)

*table-constraint-definition* ::=

UNIQUE (*column-identifier* [, *column-identifier*] ...)

| PRIMARY KEY (*column-identifier* [, *column-identifier*] ...)

| CHECK (*search-condition*)

| FOREIGN KEY *referencing-columns* REFERENCES

*ref-table-name* *referenced-columns*

## CREATE VIEW Statement

SQL conformance level: Core

CREATE VIEW *viewed-table-name*  
[( *column-identifier* [, *column-identifier*]... )]  
AS *query-specification*

## **DELETE Statement - Positioned**

**SQL conformance level: Core (ODBC 1.0)**

**SQL conformance level: Extended (ODBC 2.0)**

DELETE FROM *table-name* WHERE CURRENT OF *cursor-name*

## **DELETE Statement - Searched**

**SQL conformance level: Minimum (ODBC 2.0)**

DELETE FROM *table-name* [WHERE *search-condition*]

## **DROP INDEX Statement**

**SQL conformance level: Core**

DROP INDEX *index-name*

## **DROP TABLE Statement**

**SQL conformance level: Minimum**

DROP TABLE *base-table-name*  
[ CASCADE | RESTRICT ]

## **DROP VIEW Statement**

**SQL conformance level: Core**

DROP VIEW *viewed-table-name*  
[ CASCADE | RESTRICT ]

## GRANT Statement

SQL conformance level: Core

```
GRANT {ALL | grant-privilege [, grant-privilege]... }
ON table-name
TO {PUBLIC | user-name [, user-name]... }
```

*grant-privilege* ::=

```
DELETE
| INSERT
| SELECT
| UPDATE [( column-identifier [, column-identifier]... )]
| REFERENCES [( column-identifier
[, column-identifier]... )]
```

## INSERT Statement

SQL conformance level: Minimum

```
INSERT INTO table-name [( column-identifier [, column-identifier]...)]
VALUES (insert-value [, insert-value]... )
```

SQL conformance level: Core

```
INSERT INTO table-name [( column-identifier [, column-identifier]...)]
{ query-specification | VALUES (insert-value [, insert-value]... )}
```

## ODBC Procedure Extension

SQL conformance level: Extended

```
ODBC-std-esc-initiator [?=] call procedure ODBC-std-esc-terminator
| ODBC-ext-esc-initiator [?=] call procedure ODBC-ext-esc-terminator
```

## REVOKE Statement

**SQL conformance level: Core**

```
REVOKE {ALL | revoke-privilege [, revoke-privilege]... }  
ON table-name  
FROM {PUBLIC | user-name [, user-name]... }  
[ CASCADE | RESTRICT ]
```

*revoke-privilege* ::=

```
DELETE  
| INSERT  
| SELECT  
| UPDATE  
| REFERENCES
```

## SELECT Statement

**SQL conformance level: Minimum**

```
SELECT [ALL | DISTINCT] select-list  
FROM table-reference-list  
[WHERE search-condition]  
[order-by-clause]
```

**SQL conformance level: Core**

```
SELECT [ALL | DISTINCT] select-list  
FROM table-reference-list  
[WHERE search-condition]  
[GROUP BY column-name [, column-name]... ]  
[HAVING search-condition]  
[order-by-clause]
```

**SQL conformance level: Extended**

```

SELECT [ALL | DISTINCT] select-list
FROM table-reference-list
[WHERE search-condition]
[GROUP BY column-name [, column-name]... ]
[HAVING search-condition]
[UNION [ALL] select-statement]...
[order-by-clause]

```

(In ODBC 1.0, the UNION clause is in the SQL core grammar and does not support the ALL keyword.)

**SELECT FOR UPDATE Statement**

**SQL conformance level: Core (ODBC 1.0)**

**SQL conformance level: Extended (ODBC 2.0)**

```

SELECT [ALL | DISTINCT] select-list
FROM table-reference-list
[WHERE search-condition]
FOR UPDATE OF [column-name [, column-name]...]

```

**Statement List**

**SQL conformance level: Minimum**

```

statement ::= create-table-statement
| delete-statement-searched
| drop-table-statement
| insert-statement
| select-statement
| update-statement-searched

```

**SQL conformance level: Core**

*statement ::= alter-table-statement*  
| *create-index-statement*  
| *create-table-statement*  
| *create-view-statement*  
| *delete-statement-searched*  
| *drop-index-statement*  
| *drop-table-statement*  
| *drop-view-statement*  
| *grant-statement*  
| *insert-statement*  
| *revoke-statement*  
| *select-statement*  
| *update-statement-searched*

**SQL conformance level: Extended**

*statement ::= alter-table-statement*  
| *create-index-statement*  
| *create-table-statement*  
| *create-view-statement*  
| *delete-statement-positioned*  
| *delete-statement-searched*  
| *drop-index-statement*  
| *drop-table-statement*  
| *drop-view-statement*  
| *ODBC-procedure-extension*  
| *grant-statement*  
| *insert-statement*  
| *revoke-statement*  
| *select-statement*  
| *select-for-update-statement*  
| *statement-list*  
| *update-statement-positioned*  
| *update-statement-searched*  
  
*statement-list ::= statement | statement; statement-list*

## UPDATE Statement - Positioned

SQL conformance level: Core (ODBC 1.0)

SQL conformance level: Extended (ODBC 2.0)

```
UPDATE table-name
SET column-identifier = {expression | NULL}
[, column-identifier = {expression | NULL}]...
WHERE CURRENT OF cursor-name
```

## UPDATE Statement - Searched

SQL conformance level: Minimum

```
UPDATE table-name
SET column-identifier = {expression | NULL }
[, column-identifier = {expression | NULL}]...
[WHERE search-condition]
```

---

## Elements Used in SQL Statements

The following elements are used in the SQL statements listed previously.

---

Element	SQL Conformance Level
<i>all-function</i> ::= {AVG   MAX   MIN   SUM} ( <i>expression</i> )	Core
<i>approximate-numeric-literal</i> ::= <i>mantissa</i> <i>E</i> <i>exponent</i>	Core
<i>approximate-numeric-type</i> ::= {approximate numeric types} (For example, DOUBLE PRECISION, or REAL.)	Core
<i>argument-list</i> ::= <i>expression</i>   <i>expression</i> , <i>argument-list</i>	Minimum
<i>base-table-identifier</i> ::= <i>user-defined-name</i>	Minimum
<i>base-table-name</i> ::= <i>base-table-identifier</i>	Minimum

---

(1 of 11)

Element	SQL Conformance Level
<i>base-table-name</i> ::= <i>base-table-identifier</i>   <i>owner-name.base-table-identifier</i>   <i>qualifier-name qualifier-separator base-table-identifier</i>   <i>qualifier-name qualifier-separator [owner-name].base-table-identifier</i> (The third syntax is valid only if the data source does not support owners.)	Core
<i>between-predicate</i> ::= <i>expression</i> [NOT] BETWEEN <i>expression</i> AND <i>expression</i>	Core
<i>binary-literal</i> ::= {implementation defined}	Extended
<i>binary-type</i> ::= {binary types} (For example, LONG VARBINARY. )	Extended
<i>boolean-factor</i> ::= [NOT] <i>boolean-primary</i>	Minimum
<i>boolean-primary</i> ::= <i>predicate</i>   ( <i>search-condition</i> )	Minimum
<i>boolean-term</i> ::= <i>boolean-factor</i> [AND <i>boolean-term</i> ]	Minimum
<i>character</i> ::= {any character in the implementor's character set}	Minimum
<i>character-string-literal</i> ::= '{character}...' (To include a single literal quote character (') in a <i>character-string-literal</i> , use two literal quote characters (' ').)	Minimum
<i>character-string-type</i> ::= {character types} (For example, CHAR, VARCHAR, or LONG VARCHAR.)	Minimum
<i>column-alias</i> ::= <i>user-defined-name</i>	Core
<i>column-identifier</i> ::= <i>user-defined-name</i>	Minimum
<i>column-name</i> ::= [ <i>table-name</i> ]. <i>column-identifier</i>	Minimum
<i>column-name</i> ::= [{ <i>table-name</i>   <i>correlation-name</i> }.] <i>column-identifier</i>	Core
<i>comparison-operator</i> ::= <   >   <=   >=   =   <>	Minimum
<i>comparison-predicate</i> ::= <i>expression comparison-operator expression</i>	Minimum



Element	SQL Conformance Level
<i>comparison-predicate ::= expression comparison-operator {expression   (sub-query)}</i>	Core
<i>correlation-name ::= user-defined-name</i>	Core
<i>cursor-name ::= user-defined-name</i>	Core
<i>data-type ::= character-string-type</i>	Minimum
<i>data-type ::= character-string-type   exact-numeric-type   approximate-numeric-type</i>	Core
<i>data-type ::= character-string-type   exact-numeric-type   approximate-numeric-type   bit-type   binary-type   date-type   time-type   timestamp-type</i>	Extended
<i>date-separator ::= -</i>	Extended
<i>date-type ::= {date types}</i> (For example, DATE.)	Extended
<i>date-value ::= years-value date-separator months-value date-separator days-value</i>	Extended
<i>days-value ::= digit digit</i>	Extended
<i>digit ::= 0   1   2   3   4   5   6   7   8   9</i>	Minimum
<i>distinct-function ::= {AVG   COUNT   MAX   MIN   SUM} (DISTINCT column-name)</i>	Core
<i>dynamic-parameter ::= ?</i>	Minimum
<i>empty-string ::=</i>	Extended

(3 of 11)

Element	SQL Conformance Level
<i>escape-character</i> ::= <i>character</i>	Extended
<i>exact-numeric-literal</i> ::= [+   -] { <i>unsigned-integer</i> [. <i>unsigned-integer</i> ]   <i>unsigned-integer</i> .   <i>.unsigned-integer</i> }	Core
<i>exact-numeric-type</i> ::= {exact numeric types} (For example, DECIMAL, SMALLINT, or INTEGER.)	Core
<i>exists-predicate</i> ::= EXISTS ( <i>sub-query</i> )	Core
<i>exponent</i> ::= [+   -] <i>unsigned-integer</i>	Core
<i>expression</i> ::= <i>term</i>   <i>expression</i> {+   -} <i>term</i>	Minimum
<i>factor</i> ::= [+   -] <i>primary</i>	Minimum
<i>hours-value</i> ::= <i>digit digit</i>	Extended
<i>index-identifier</i> ::= <i>user-defined-name</i>	Core
<i>index-name</i> ::= [ <i>index-qualifier</i> .] <i>index-identifier</i>	Core
<i>index-qualifier</i> ::= <i>user-defined-name</i>	Core
<i>in-predicate</i> ::= <i>expression</i> [NOT] IN {(value {, value}...)   ( <i>sub-query</i> )}	Core
<i>insert-value</i> ::= <i>dynamic-parameter</i>   <i>literal</i>   NULL   USER	Minimum
<i>keyword</i> ::= (see list of reserved keywords)	Minimum
<i>length</i> ::= <i>unsigned-integer</i>	Minimum
<i>letter</i> ::= <i>lower-case-letter</i>   <i>upper-case-letter</i>	Minimum
<i>like-predicate</i> ::= <i>expression</i> [NOT] LIKE <i>pattern-value</i>	Minimum

Element	SQL Conformance Level
<i>like-predicate</i> ::= <i>expression</i> [NOT] LIKE <i>pattern-value</i> [ <i>ODBC-like-escape-clause</i> ]	Extended
<i>literal</i> ::= <i>character-string-literal</i>	Minimum
<i>literal</i> ::= <i>character-string-literal</i>   <i>numeric-literal</i>	Core
<i>literal</i> ::= <i>character-string-literal</i>   <i>numeric-literal</i>   <i>bit-literal</i>   <i>binary-literal</i>   <i>ODBC-date-time-extension</i>	Extended
<i>lower-case-letter</i> ::= a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z	Minimum
<i>mantissa</i> ::= <i>exact-numeric-literal</i>	Core
<i>minutes-value</i> ::= <i>digit digit</i>	Extended
<i>months-value</i> ::= <i>digit digit</i>	Extended
<i>null-predicate</i> ::= <i>column-name</i> IS [NOT] NULL	Minimum
<i>numeric-literal</i> ::= <i>exact-numeric-literal</i>   <i>approximate-numeric-literal</i>	Minimum
<i>ODBC-date-literal</i> ::= <i>ODBC-std-esc-initiator</i> d ' <i>date-value</i> ' <i>ODBC-std-esc-terminator</i>   <i>ODBC-ext-esc-initiator</i> d ' <i>date-value</i> ' <i>ODBC-ext-esc-terminator</i>	Extended
<i>ODBC-date-time-extension</i> ::= <i>ODBC-date-literal</i>   <i>ODBC-time-literal</i>   <i>ODBC-timestamp-literal</i>	Extended
<i>ODBC-like-escape-clause</i> ::= <i>ODBC-std-esc-initiator</i> escape ' <i>escape-character</i> ' <i>ODBC-std-esc-terminator</i>   <i>ODBC-ext-esc-initiator</i> escape ' <i>escape-character</i> ' <i>ODBC-ext-esc-terminator</i>	Extended

Element	SQL Conformance Level
<i>ODBC-time-literal</i> ::=   <i>ODBC-std-esc-initiator</i> t 'time-value' <i>ODBC-std-esc-terminator</i>   <i>ODBC-ext-esc-initiator</i> t 'time-value' <i>ODBC-ext-esc-terminator</i>	Extended
<i>ODBC-timestamp-literal</i> ::=   <i>ODBC-std-esc-initiator</i> ts 'timestamp-value' <i>ODBC-std-esc-terminator</i>   <i>ODBC-ext-esc-initiator</i> ts 'timestamp-value' <i>ODBC-ext-esc-terminator</i>	Extended
<i>ODBC-ext-esc-initiator</i> ::= {	Extended
<i>ODBC-ext-esc-terminator</i> ::= }	Extended
<i>ODBC-outer-join-extension</i> ::= <i>ODBC-std-esc-initiator</i> oj <i>outer-join</i> <i>ODBC-std-esc-terminator</i>   <i>ODBC-ext-esc-initiator</i> oj <i>outer-join</i> <i>ODBC-ext-esc-terminator</i>	Extended
<i>ODBC-scalar-function-extension</i> ::= <i>ODBC-std-esc-initiator</i> fn <i>scalar-function</i> <i>ODBC-std-esc-terminator</i>   <i>ODBC-ext-esc-initiator</i> fn <i>scalar-function</i> <i>ODBC-ext-esc-terminator</i>	Extended
<i>ODBC-std-esc-initiator</i> ::= <i>ODBC-std-esc-prefix</i> <i>SQL-esc-vendor-clause</i>	Extended
<i>ODBC-std-esc-prefix</i> ::= --(*	Extended
<i>ODBC-std-esc-terminator</i> ::= *)--	Extended
<i>order-by-clause</i> ::= ORDER BY <i>sort-specification</i> [, <i>sort-specification</i> ]...	Minimum
<i>outer-join</i> ::= <i>table-name</i> [ <i>correlation-name</i> ] LEFT OUTER JOIN { <i>table-name</i> [ <i>correlation-name</i> ]   <i>outer-join</i> } ON <i>search-condition</i> (For outer joins, <i>search-condition</i> must contain only the join condition between the specified <i>table-names</i> .)	Extended
<i>owner-name</i> ::= <i>user-defined-name</i>	Core
<i>pattern-value</i> ::= <i>character-string-literal</i>   <i>dynamic-parameter</i> (In a <i>character-string-literal</i> , the percent character (%) matches 0 or more of any character; the underscore character (_) matches 1 character.)	Minimum

Element	SQL Conformance Level
<i>pattern-value ::= character-string-literal   dynamic-parameter   USER</i> (In a <i>character-string-literal</i> , the percent character (%) matches 0 or more of any character; the underscore character (_) matches 1 character.)	Core
<i>precision ::= unsigned-integer</i>	Core
<i>predicate ::= comparison-predicate   like-predicate   null-predicate</i>	Minimum
<i>predicate ::=</i> <i>between-predicate   comparison-predicate   exists-predicate</i> <i>  in-predicate   like-predicate   null-predicate   quantified-predicate</i>	Core
<i>primary ::= column-name</i> <i>  dynamic-parameter</i> <i>  literal</i> <i>  ( expression )</i>	Minimum
<i>primary ::= column-name</i> <i>  dynamic-parameter</i> <i>  literal</i> <i>  set-function-reference</i> <i>  USER</i> <i>  ( expression )</i>	Core
<i>primary ::= column-name</i> <i>  dynamic-parameter</i> <i>  literal</i> <i>  ODBC-scalar-function-extension</i> <i>  set-function-reference</i> <i>  USER</i> <i>  ( expression )</i>	Extended
<i>procedure ::= procedure-name   procedure-name (procedure-parameter-list)</i>	Extended
<i>procedure-identifier ::= user-defined-name</i>	Extended

(7 of 11)

Element	SQL Conformance Level
<p><i>procedure-name</i> ::= <i>procedure-identifier</i>    <i>owner-name.procedure-identifier</i>    <i>qualifier-name qualifier-separator procedure-identifier</i>    <i>qualifier-name qualifier-separator [owner-name].procedure-identifier</i>  (The third syntax is valid only if the data source does not support owners.)</p>	Extended
<p><i>procedure-parameter-list</i> ::= <i>procedure-parameter</i>    <i>procedure-parameter, procedure-parameter-list</i></p>	Extended
<p><i>procedure-parameter</i> ::= <i>dynamic-parameter</i>   <i>literal</i>   <i>empty-string</i>  (If a procedure parameter is an empty string, the procedure uses the default value for that parameter.)</p>	Extended
<p><i>qualifier-name</i> ::= <i>user-defined-name</i></p>	Core
<p><i>qualifier-separator</i> ::= {implementation-defined}  (The qualifier separator is returned through <b>SQLGetInfo</b> with the <code>SQL_QUALIFIER_NAME_SEPARATOR</code> option.)</p>	Core
<p><i>quantified-predicate</i> ::= <i>expression comparison-operator</i> {<i>ALL</i>   <i>ANY</i>}  (<i>sub-query</i>)</p>	Core
<p><i>query-specification</i> ::=  SELECT [ALL   DISTINCT]] <i>select-list</i>  FROM <i>table-reference-list</i>  [WHERE <i>search-condition</i>]  [GROUP BY <i>column-name</i>, [<i>column-name</i>]...]  [HAVING <i>search-condition</i>]</p>	Core
<p><i>ref-table-name</i> ::= <i>base-table-identifier</i></p>	Minimum
<p><i>ref-table-name</i> ::= <i>base-table-identifier</i>    <i>owner-name.base-table-identifier</i>    <i>qualifier-name qualifier-separator base-table-identifier</i>    <i>qualifier-name qualifier-separator [owner-name].base-table-identifier</i>  (The third syntax is valid only if the data source does not support owners.)</p>	Core
<p><i>referenced-columns</i> ::= ( <i>column-identifier</i> [, <i>column-identifier</i>]... )</p>	Core
<p><i>referencing-columns</i> ::= ( <i>column-identifier</i> [, <i>column-identifier</i>]... )</p>	Core

Element	SQL Conformance Level
<i>scalar-function</i> ::= <i>function-name</i> ( <i>argument-list</i> ) (The definitions for the non-terminals <i>function-name</i> and <i>function-name</i> ( <i>argument-list</i> ) are derived from the list of scalar functions in Appendix F, “Scalar Functions.”)	Extended
<i>scale</i> ::= <i>unsigned-integer</i>	Core
<i>search-condition</i> ::= <i>boolean-term</i> [OR <i>search-condition</i> ]	Minimum
<i>seconds-fraction</i> ::= <i>unsigned-integer</i>	Extended
<i>seconds-value</i> ::= <i>digit digit</i>	Extended
<i>select-list</i> ::= *   <i>select-sublist</i> [, <i>select-sublist</i> ]...	Minimum
<i>select-sublist</i> ::= <i>expression</i>	Minimum
<i>select-sublist</i> ::= <i>expression</i> [[AS] <i>column-alias</i> ]   { <i>table-name</i>   <i>correlation-name</i> }.*	Core
<i>set-function-reference</i> ::= COUNT(*)   <i>distinct-function</i>   <i>all-function</i>	Core
<i>sort-specification</i> ::= { <i>unsigned-integer</i>   <i>column-name</i> } [ASC   DESC]	Minimum
<i>SQL-esc-vendor-clause</i> ::= VENDOR(Microsoft), PRODUCT(ODBC)	Extended
<i>sub-query</i> ::= SELECT [ALL   DISTINCT] <i>select-list</i> FROM <i>table-reference-list</i> [WHERE <i>search-condition</i> ] [GROUP BY <i>column-name</i> [, <i>column-name</i> ]...] [HAVING <i>search-condition</i> ]	Core
<i>table-identifier</i> ::= <i>user-defined-name</i>	Minimum
<i>table-name</i> ::= <i>table-identifier</i>	Minimum
<i>table-name</i> ::= <i>table-identifier</i>   <i>owner-name.table-identifier</i>   <i>qualifier-name qualifier-separator table-identifier</i>   <i>qualifier-name qualifier-separator</i> [ <i>owner-name</i> ]. <i>table-identifier</i> (The third syntax is valid only if the data source does not support owners.)	Core

Element	SQL Conformance Level
<i>table-reference</i> ::= <i>table-name</i>	Minimum
<i>table-reference</i> ::= <i>table-name</i> [ <i>correlation-name</i> ]	Core
<i>table-reference</i> ::= <i>table-name</i> [ <i>correlation-name</i> ]   <i>ODBC-outer-join-extension</i> (A SELECT statement can contain only one <i>table-reference</i> that is an <i>ODBC-outer-join-extension</i> .)	Extended
<i>table-reference-list</i> ::= <i>table-reference</i> [, <i>table-reference</i> ]...	Core
<i>term</i> ::= <i>factor</i>   <i>term</i> { *   / } <i>factor</i>	Minimum
<i>timestamp-separator</i> ::= (The blank character.)	Extended
<i>timestamp-type</i> ::= {timestamp types} (For example, TIMESTAMP. To determine the type name used by a data source, an application calls <b>SQLGetTypeInfo</b> .)	Extended
<i>timestamp-value</i> ::= <i>date-value</i> <i>timestamp-separator</i> <i>time-value</i> [, <i>seconds-fraction</i> ]	Extended
<i>unsigned-integer</i> ::= { <i>digit</i> }...	Minimum
<i>upper-case-letter</i> ::= A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z	Minimum
<i>user-defined-name</i> ::= <i>letter</i> [ <i>digit</i>   <i>letter</i>   _]...	Minimum
<i>user-name</i> ::= <i>user-defined-name</i>	Core
<i>value</i> ::= <i>literal</i>   USER   <i>dynamic-parameter</i>	Core



Element	SQL Conformance Level
<i>viewed-table-identifier ::= user-defined-name</i>	Core
<i>viewed-table-name ::= viewed-table-identifier</i>   <i>owner-name.viewed-table-identifier</i>   <i>qualifier-name qualifier-separator viewed-table-identifier</i>   <i>qualifier-name qualifier-separator [owner-name].viewed-table-identifier</i> (The third syntax is valid only if the data source does not support owners.)	Core
<i>years-value ::= digit digit digit digit</i>	Extended

(11 of 11)

## List of Reserved Keywords

The following words are reserved for use in ODBC function calls. These words do not constrain the minimum SQL grammar; however, to ensure compatibility with drivers that support the core SQL grammar, applications should avoid using any of these keywords. The **#define** value `SQL_ODBC_KEYWORDS` contains a comma-separated list of these keywords.

- ABSOLUTE
- ADA
- ADD
- ALL
- ALLOCATE
- ALTER
- AND
- ANY
- ARE
- AS
- ASC
- ASSERTION

## *List of Reserved Keywords*

- AT
- AUTHORIZATION
- AVG
- BEGIN
- BETWEEN
- BIT
- BIT\_LENGTH
- BY
- CASCADE
- CASCADED
- CASE
- CAST
- CATALOG
- CHAR
- CHAR\_LENGTH
- CHARACTER
- CHARACTER\_LENGTH
- CHECK
- CLOSE
- COALESCE
- COBOL
- COLLATE
- COLLATION
- COLUMN
- COMMIT
- CONNECT
- CONNECTION
- CONSTRAINT
- CONSTRAINTS
- CONTINUE
- CONVERT
- CORRESPONDING
- COUNT

- CREATE
- CURRENT
- CURRENT\_DATE
- CURRENT\_TIME
- CURRENT\_TIMESTAMP
- CURSOR
- DATE
- DAY
- DEALLOCATE
- DEC
- DECIMAL
- DECLARE
- DEFERRABLE
- DEFERRED
- DELETE
- DESC
- DESCRIBE
- DESCRIPTOR
- DIAGNOSTICS
- DICTIONARY
- DISCONNECT
- DISPLACEMENT
- DISTINCT
- DOMAIN
- DOUBLE
- DROP
- ELSE
- END
- END-EXEC
- ESCAPE
- EXCEPT
- EXCEPTION
- EXEC

## *List of Reserved Keywords*

- EXECUTE
- EXISTS
- EXTERNAL
- EXTRACT
- FALSE
- FETCH
- FIRST
- FLOAT
- FOR
- FOREIGN
- FORTRAN
- FOUND
- FROM
- FULL
- GET
- GLOBAL
- GO
- GOTO
- GRANT
- GROUP
- HAVING
- HOUR
- IDENTITY
- IGNORE
- IMMEDIATE
- IN
- INCLUDE
- INDEX
- INDICATOR
- INITIALLY
- INNER
- INPUT
- INSENSITIVE

- INSERT
- INTEGER
- INTERSECT
- INTERVAL
- INTO
- IS
- ISOLATION
- JOIN
- KEY
- LANGUAGE
- LAST
- LEFT
- LEVEL
- LIKE
- LOCAL
- LOWER
- MATCH
- MAX
- MIN
- MINUTE
- MODULE
- MONTH
- MUMPS
- NAMES
- NATIONAL
- NCHAR
- NEXT
- NONE
- NOT
- NULL
- NULLIF
- NUMERIC
- OCTET\_LENGTH

## *List of Reserved Keywords*

- OF
- OFF
- ON
- ONLY
- OPEN
- OPTION
- OR
- ORDER
- OUTER
- OUTPUT
- OVERLAPS
- PARTIAL
- PASCAL
- PLI
- POSITION
- PRECISION
- PREPARE
- PRESERVE
- PRIMARY
- PRIOR
- PRIVILEGES
- PROCEDURE
- PUBLIC
- RESTRICT
- REVOKE
- RIGHT
- ROLLBACK
- ROWS
- SCHEMA
- SCROLL
- SECOND
- SECTION
- SELECT

- SEQUENCE
- SET
- SIZE
- SMALLINT
- SOME
- SQL
- SQLCA
- SQLCODE
- SQLERROR
- SQLSTATE
- SQLWARNING
- SUBSTRING
- SUM
- SYSTEM
- TABLE
- TEMPORARY
- THEN
- TIME
- TIMESTAMP
- TIMEZONE\_HOUR
- TIMEZONE\_MINUTE
- TO
- TRANSACTION
- TRANSLATE
- TRANSLATION
- TRUE
- UNION
- UNIQUE
- UNKNOWN
- UPDATE
- UPPER
- USAGE
- USER

## *List of Reserved Keywords*

- USING
- VALUE
- VALUES
- VARCHAR
- VARYING
- VIEW
- WHEN
- WHENEVER
- WHERE
- WITH
- WORK
- YEAR



# Data Types

Data stored on a data source has an SQL data type, which might be specific to that data source. The Informix driver maps data source-specific SQL data types to ODBC SQL data types. The driver returns these mappings through **SQLGetTypeInfo**. It also returns the SQL data types when describing the data types of columns and parameters in **SQLColAttributes**, **SQLColumns**, **SQLDescribeCol**, and **SQLSpecialColumns**.

Each ODBC SQL data type corresponds to an ODBC C data type. By default, the Informix driver assumes that the C data type of a storage location corresponds to the SQL data type of the column or parameter to which the location is bound. If the C data type of a storage location is not the *default* C data type, the application can specify the correct C data type with the *fcType* argument in **SQLGetData** or in **SQLBindParameter**. Before returning data from the data source, the Informix driver converts it to the specified C data type; before sending data to the data source, the Informix driver converts it from the specified C data type.

This appendix discusses the following topics:

- ODBC SQL data types that INFORMIX-CLI supports
- ODBC C data types
- Default ODBC C data types
- Transferring data in its binary form
- Precision, scale, length, and display size of SQL data types
- Converting data from SQL to C data types
- Converting data from C to SQL data types

## ODBC SQL Data Types That INFORMIX-CLI Supports

The following table lists valid values of *fSqlType* for the SQL data types that INFORMIX-CLI supports. In the table, “precision” refers to the total number of digits, and “scale” refers to the number of digits to the right of the decimal point.

To determine which data types are supported by a data source and the characteristics of those data types, an application calls **SQLGetTypeInfo**. For information about how Informix data types map to ODBC SQL data types, see, “[Mapping Data Types](#)” on page 1-15. For information on Informix data types, refer to the [Informix Guide to SQL: Reference](#).

fSqlType	SQL Data Type	Description
SQL_CHAR	CHAR( <i>n</i> )	Character string of fixed string length <i>n</i> ( $1 \leq n \leq 254$ )
SQL_VARCHAR	VARCHAR( <i>n</i> )	Variable-length character string with a maximum string length <i>n</i> ( $1 \leq n \leq 254$ )
SQL_LONGVARCHAR	LONG VARCHAR	Variable length character data. Maximum length is data source-dependent
SQL_DECIMAL	DECIMAL( <i>p,s</i> )	Signed, exact, numeric value with a precision <i>p</i> and scale <i>s</i> ( $1 \leq p \leq 15$ ; $0 \leq s \leq p$ )
SQL_SMALLINT	SMALLINT	Exact numeric value with precision 5 and scale 0 (signed: $-32,768 \leq n \leq 32,767$ , unsigned: $0 \leq n \leq 65,535$ ) <sup>a</sup>
SQL_INTEGER	INTEGER	Exact numeric value with precision 10 and scale 0 (signed: $-2^{31} \leq n \leq 2^{31} - 1$ , unsigned: $0 \leq n \leq 2^{32} - 1$ ) <sup>a</sup>

(1 of 2)

fSqlType	SQL Data Type	Description
SQL_REAL	REAL	Signed, approximate, numeric value with a mantissa precision 7 (zero or absolute value $10^{-38}$ to $10^{38}$ )
SQL_DOUBLE	DOUBLE PRECISION	Signed, approximate, numeric value with a mantissa precision 15 (zero or absolute value $10^{-308}$ to $10^{308}$ )
SQL_LONGVARBINARY	LONG VARBINARY	Variable length binary data. Maximum length is data source-dependent.
SQL_DATE	DATE	Date data
SQL_TIMESTAMP	TIMESTAMP	Date/time data
<p><sup>a</sup> An application uses <b>SQLGetTypeInfo</b> or <b>SQLColAttributes</b> to determine if a particular data type or a particular column in a result set is unsigned.</p>		

(2 of 2)

## C Data Types

Data is stored in the application in ODBC C data types. The core C data types support the minimum and core SQL data types. They also support some extended SQL data types. The extended C data types support only extended SQL data types.

The C data type is specified in the **SQLBindCol**, **SQLGetData**, and **SQLBindParameter** functions with the *fctype* argument.

### Core C Data Types

The following table lists valid values of *fctype* for the core C data types. These values are defined in **sql.h**. The table also lists the ODBC C data type that implements each value of *fctype* and the definition of this data type from **sql.h**.

fctype	ODBC C Typedef	C Type
SQL_C_CHAR	UCHAR FAR *	unsigned char FAR *
SQL_C_SSHORT	SWORD	short int
SQL_C_USHORT	UWORD	unsigned short int
SQL_C_SLONG	SDWORD	long int
SQL_C_ULONG	UDWORD	unsigned long int
SQL_C_FLOAT	SFLOAT	float
SQL_C_DOUBLE	SDOUBLE	double



**Important:** Because string arguments in *INFORMIX-CLI* functions are unsigned, applications that pass *CString* objects to *INFORMIX-CLI* functions without casting them to unsigned strings receive compiler warnings.

## Extended C Data Types

The following table lists valid values of *fCType* for the extended C data types. These values are defined in **sqlext.h**. The table also lists the ODBC C data type that implements each value of *fCType* and the definition of this data type from **sqlext.h** or **sql.h**.

<b>fCType</b>	<b>ODBC C Typedef</b>	<b>C Type</b>
SQL_C_BIT	UCHAR	unsigned char
SQL_C_STINYINT	SCHAR	signed char
SQL_C_UTINYINT	UCHAR	unsigned char
SQL_C_BINARY	UCHAR FAR *	unsigned char FAR *
SQL_C_DATE	DATE_STRUCT	struct tagDATE_STRUCT { SWORD year; <sup>a</sup> UWORD month; <sup>b</sup> UWORD day; <sup>c</sup> }
SQL_C_TIME	TIME_STRUCT	struct tagTIME_STRUCT { UWORD hour; <sup>d</sup> UWORD minute; <sup>e</sup> UWORD second; <sup>f</sup> }
SQL_C_TIMESTAMP	TIMESTAMP_STRUCT	struct tagTIMESTAMP_STRUCT { SWORD year; <sup>a</sup> UWORD month; <sup>b</sup> UWORD day; <sup>c</sup> UWORD hour; <sup>d</sup> UWORD minute; <sup>e</sup> UWORD second; <sup>f</sup> UDWORD fraction; <sup>g</sup> }

<sup>a</sup> The value of the year field must be in the range from 0 to 9,999. Years are measured from 0 A.D. Some data sources do not support the entire range of years.

<sup>b</sup> The value of the month field must be in the range from 1 to 12.

(1 of 2)

fCType	ODBC C Typedef	C Type
<sup>c</sup> The value of the day field must be in the range from 1 to the number of days in the month. The number of days in the month is determined from the values of the year and month fields and is 28, 29, 30, or 31.		
<sup>d</sup> The value of the hour field must be in the range from 0 to 23.		
<sup>e</sup> The value of the minute field must be in the range from 0 to 59.		
<sup>f</sup> The value of the second field must be in the range from 0 to 59.		
<sup>g</sup> The value of the fraction field is the number of nanoseconds ( $10^{-9}$ seconds) and ranges from 0 to 999,999,999. For example, the value of the fraction field for a half-second is 500,000,000, for a thousandth of a second (one millisecond) is 1,000,000, for a millionth of a second (one microsecond) is 1,000, and for a billionth of a second (one nanosecond) is 1.		

(2 of 2)

## Default C Data Types

If an application specifies `SQL_C_DEFAULT` for the *fCType* argument in `SQLBindCol`, `SQLGetData`, or `SQLBindParameter`, the driver assumes that the C data type of the output or input buffer corresponds to the SQL data type of the column or parameter to which the buffer is bound. For each ODBC SQL data type, the following table shows the corresponding, or *default*, C data type. For information about how Informix data types map to ODBC SQL data types, see, [“Mapping Data Types” on page 1-15](#)



**Tip:** For maximum interoperability, applications should specify a C data type other than `SQL_C_DEFAULT`. This allows drivers that promote SQL data types (and therefore cannot always determine default C data types) to return data. It also allows drivers that cannot determine whether an integer column is signed or unsigned to correctly return data.



**Important:** ODBC 2.0 drivers use the ODBC 2.0 default C data types for ODBC 1.0 and ODBC 2.0 integer C data.

SQL Data Type	Default C Data Type
SQL_CHAR	SQL_C_CHAR
SQL_VARCHAR	SQL_C_CHAR
SQL_LONGVARCHAR	SQL_C_CHAR
SQL_DECIMAL	SQL_C_CHAR
SQL_SMALLINT	SQL_C_SSHORT or SQL_C_USHORT <sup>a</sup>
SQL_INTEGER	SQL_C_SLONG or SQL_C_ULONG <sup>a</sup>
SQL_REAL	SQL_C_FLOAT
SQL_DOUBLE	SQL_C_DOUBLE
SQL_LONGVARBINARY	SQL_C_BINARY
SQL_DATE	SQL_C_DATE
SQL_TIMESTAMP	SQL_C_TIMESTAMP

<sup>a</sup> If the driver can determine whether the column is signed or unsigned, such as when the driver is fetching data from the data source or when the data source supports only a signed type or only an unsigned type, but not both, the driver uses the corresponding signed or unsigned C data type. If the driver cannot determine whether the column is signed or unsigned, it passes the data value without attempting to validate it numerically.

---

## Transferring Data in its Binary Form

Among data sources that use the same DBMS, an application can safely transfer data in the internal form used by that DBMS. For a particular piece of data, the SQL data types must be the same in the source and target data sources. The C data type is `SQL_C_BINARY`.

When the application calls `SQLFetch`, `SQLExtendedFetch`, or `SQLGetData` to retrieve the data from the source data source, the driver retrieves the data from the data source and transfers it, without conversion, to a storage location of type `SQL_C_BINARY`. When the application calls `SQLExecute`, `SQLExecDirect`, or `SQLPutData` to send the data to the target data source, the driver retrieves the data from the storage location and transfers it, without conversion, to the target data source.

***Important:** Applications that transfer any data (except binary data) in this manner are not interoperable among DBMSs.*



---

## Precision, Scale, Length, and Display Size

`SQLColAttributes`, `SQLColumns`, and `SQLDescribeCol` return the precision, scale, length, and display size of a column in a table. `SQLProcedureColumns` returns the precision, scale, and length of a column in a procedure.

`SQLDescribeParam` returns the precision or scale of a parameter in an SQL statement; `SQLBindParameter` sets the precision or scale of a parameter in an SQL statement. `SQLGetTypeInfo` returns the maximum precision and the minimum and maximum scales of an SQL data type on a data source.

`SQLColAttributes`, `SQLColumns`, and `SQLDescribeCol` return the precision, scale, length, and display size of a column in a table. `SQLBindParameter` sets the precision or scale of a parameter in an SQL statement. `SQLGetTypeInfo` returns the maximum precision and the minimum and maximum scales of an SQL data type on a data source.

Because of limitations in the size of the arguments that these functions use, precision, length, and display size are limited to the size of an `SDWORD`, or 2,147,483,647.



## Precision

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 nonnumeric column or parameter generally refers to either the maximum length or the defined length of the column or parameter. To determine the maximum precision allowed for a data type, an application calls **SQLGetTypeInfo**. The following table defines the precision for each ODBC SQL data type.

fSqlType	Precision
SQL_CHAR SQL_VARCHAR	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR <sup>a, b</sup>	The maximum length of the column or parameter.
SQL_DECIMAL	The defined number of digits. For example, the precision of a column defined as NUMERIC(10,3) is 10.
SQL_SMALLINT <sup>c</sup>	5
SQL_INTEGER <sup>c</sup>	10
SQL_REAL <sup>c</sup>	7
SQL_DOUBLE <sup>c</sup>	15
SQL_LONGVARBINARY <sup>a, b</sup>	The maximum length of the column or parameter.
SQL_DATE <sup>c</sup>	10 (the number of characters in the yyyy-mm-dd format).
SQL_TIMESTAMP	The number of characters in the “yyyy-mm-dd hh:mm:ss[.f...]” format used by the TIMESTAMP data type. For example, if a time stamp 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 time stamp uses thousandths of a second, the precision is 23 (the number of characters in the “yyyy-mm-dd hh:mm:ss.fff” format).

(1 of 2)

fSqlType	Precision
	<sup>a</sup> For an ODBC 1.0 application calling <b>SQLSetParam</b> in an ODBC 2.0 driver, when <i>pcbValue</i> is SQL_DATA_AT_EXEC, <i>cbColDef</i> must be set to the total length of the data to be sent, not the precision as defined in this table.
	<sup>b</sup> If the driver cannot determine the column or parameter length, it returns SQL_NO_TOTAL.
	<sup>c</sup> The <i>cbColDef</i> argument of <b>SQLBindParameter</b> is ignored for this data type.

(2 of 2)

## Scale

The scale of a numeric column or parameter refers to the maximum number of digits to the right of the decimal point. For approximate floating point number columns or parameters, the scale is undefined because the number of digits to the right of the decimal point is not fixed. (For the SQL\_DECIMAL and SQL\_NUMERIC data types, the maximum scale is generally the same as the maximum precision. However, some data sources impose a separate limit on the maximum scale. To determine the minimum and maximum scales allowed for a data type, an application calls **SQLGetTypeInfo**.) The following table defines the scale for each ODBC SQL data type.

fSqlType	Scale
SQL_CHAR <sup>a</sup> SQL_VARCHAR <sup>a</sup> SQL_LONGVARCHAR <sup>a</sup>	Not applicable.
SQL_DECIMAL	The defined number of digits to the right of the decimal point. For example, the scale of a column defined as NUMERIC(10,3) is 3.
SQL_SMALLINT <sup>a</sup> SQL_INTEGER <sup>a</sup>	0
SQL_REAL <sup>a</sup> SQL_DOUBLE <sup>a</sup>	Not applicable.
SQL_LONGVARBINARY <sup>a</sup>	Not applicable.

(1 of 2)

fSqlType	Scale
SQL_DATE <sup>a</sup>	Not applicable.
SQL_TIMESTAMP	The number of digits to the right of the decimal point in the “yyyy-mm-dd hh:mm:ss[.f...]” format. For example, if the TIMESTAMP data type uses the “yyyy-mm-dd hh:mm:ss.fff” format, the scale is 3.

<sup>a</sup> The *ibScale* argument of **SQLBindParameter** is ignored for this data type.

(2 of 2)

## Length

The length of a column is the maximum number of bytes that are 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. The length of a column might be different than the number of bytes that are required to store the data on the data source. For a list of default C data types, see the [“Default C Data Types” on page C-6](#).

The following table defines the length for each ODBC SQL data type.

fSqlType	Length
SQL_CHAR SQL_VARCHAR	The defined length of the column. For example, the length of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR <sup>a</sup>	The maximum length of the column.
SQL_DECIMAL	The maximum number of digits plus 2. Because 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_SMALLINT	2 (2 bytes).
SQL_INTEGER	4 (4 bytes).
SQL_REAL	4 (4 bytes).

(1 of 2)

fSqlType	Length
SQL_DOUBLE	8 (8 bytes).
SQL_LONGVARBINARY <sup>a</sup>	The maximum length of the column.
SQL_DATE	6 (the size of the DATE_STRUCT or TIME_STRUCT structure).
SQL_TIMESTAMP	16 (the size of the TIMESTAMP_STRUCT structure).
<sup>a</sup> If the driver cannot determine the column or parameter length, it returns SQL_NO_TOTAL.	

(2 of 2)

## Display Size

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 ODBC SQL data type.

fSqlType	Display Size
SQL_CHAR SQL_VARCHAR	The defined length of the column. For example, the display size of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR <sup>a</sup>	The maximum length of the column.
SQL_DECIMAL	The precision of the column plus 2 (a sign, <i>precision</i> digits, and a decimal point). For example, the display size of a column defined as NUMERIC(10,3) is 12.
SQL_SMALLINT	6 if signed (a sign and 5 digits) or 5 if unsigned (5 digits).
SQL_INTEGER	11 if signed (a sign and 10 digits) or 10 if unsigned (10 digits).
SQL_REAL	13 (a sign, 7 digits, a decimal point, the letter E, a sign, and 2 digits).
SQL_DOUBLE	22 (a sign, 15 digits, a decimal point, the letter E, a sign, and 3 digits).

(1 of 2)

fSqlType	Display Size
SQL_LONGVARBINARY <sup>a</sup>	The maximum length of the column times 2.
SQL_DATE	10 (a date in the format yyyy-mm-dd).
SQL_TIMESTAMP	19 (if the scale of the time stamp is 0) or 20 plus the scale of the time stamp (if the scale is greater than 0). This is the number of characters in the “yyyy-mm-dd hh:mm:ss[.f...]” 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”).

<sup>a</sup> If the driver cannot determine the column or parameter length, it returns SQL\_NO\_TOTAL.

(2 of 2)

## Converting Data from SQL to C Data Types

When an application calls **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData**, the driver retrieves the data from the data source. If necessary, it converts the data from the data type in which the driver retrieved it to the data type specified by the *fCType* argument in **SQLBindCol** or **SQLGetData**. Finally, it stores the data in the location pointed to by the *rgbValue* argument in **SQLBindCol** or **SQLGetData**.

The word *convert* is used in this section in a broad sense, and it includes the transfer of data, without a conversion in data type, from one storage location to another.

The following table shows the supported conversions from ODBC SQL data types to ODBC C data types. A solid circle (●) indicates the default conversion for an SQL data type (the C data type to which the data is converted when the value of *fCType* is SQL\_C\_DEFAULT). A hollow circle (○) indicates a supported conversion.

## Converting Data from SQL to C Data Types

SQL Data Type	C Data Type																	
	SQL_C_CHAR	SQL_C_BIT	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_TINYINT	SQL_C_SSHORT	SQL_C_USHORT	SQL_C_SHORT	SQL_C_SLONG	SQL_C_ULONG	SQL_C_LONG	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_BINARY	SQL_C_DATE	SQL_C_TIME	SQL_C_TIMESTAMP	
SQL_CHAR	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_VARCHAR	●●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_LONGVARCHAR	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_DECIMAL	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_SMALLINT (signed)	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○
SQL_SMALLINT	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○
SQL_INTEGER (signed)	○○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○
SQL_INTEGER (unsigned)	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○
SQL_REAL	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○
SQL_DOUBLE	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○
SQL_LONGVARBINARY	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
SQL_DATE	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○
SQL_TIMESTAMP	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●

The tables in the following sections describe how the driver or data source converts data that is retrieved from the data source; drivers are required to support conversions to all ODBC C data types from the ODBC SQL data types that they support. For a given ODBC 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. For each outcome, the third and fourth columns list the values of the *rgbValue* and *pcbValue* arguments specified in **SQLBindCol** or **SQLGetData** after the driver attempts to convert the data.

The last column lists the SQLSTATE returned for each outcome by **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData**.

If the *fctype* argument in **SQLBindCol** or **SQLGetData** contains a value for an ODBC C data type that is not shown in the table for a given ODBC SQL data type, **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData** returns SQLSTATE 07006 (Restricted data type attribute violation). If the *fctype* argument contains a value that specifies a conversion from a driver-specific SQL data type to an ODBC C data type and this conversion is not supported by the driver, **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData** returns SQLSTATE S1C00 (Driver not capable).

Although it is not shown in the tables in this appendix, 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 “Usage” on page 13-163. 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 S1009 (Invalid argument value).

The following terms and conventions are used in the tables:

- *Length of data* is the number of bytes of C data that are available to return in *rgbValue*, regardless of whether the data is truncated before it returns to the application. For string data, this does not include the null-termination byte.
- *Display size* is the total number of bytes that are needed to display the data in character format.
- Words in *italics* represent function arguments or elements of the ODBC SQL grammar. For the syntax of grammar elements, see [Appendix B](#).

## SQL to C: Character

The character ODBC SQL data types are found in the following list:

- SQL\_CHAR
- SQL\_VARCHAR
- SQL\_LONGVARCHAR

The following table shows the ODBC C data types to which character SQL data can be converted. For an explanation of the columns and terms in the table, see the previous list.

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_CHAR	Length of data < <i>cbValueMax</i>	Data	Length of data	N/A
	Length of data ≥ <i>cbValueMax</i>	Truncated data	Length of data	01004
SQL_C_STINYINT SQL_C_UTINYINT SQL_C_TINYINT <sup>a</sup>	Data converted without truncation <sup>b</sup>	Data	Size of the C data type	N/A
SQL_C_SSHORT SQL_C_USHORT SQL_C_SHORT <sup>a</sup>	Data converted with truncation of fractional digits <sup>b</sup>	Truncated data	Size of the C data type	01004
SQL_C_SLONG SQL_C_ULONG SQL_C_LONG <sup>a</sup>	Conversion of data would result in loss of whole (as opposed to fractional) digits <sup>b</sup>	Untouched	Untouched	22003
	Data is not a <i>numeric-literal</i> <sup>b</sup>	Untouched	Untouched	22005
SQL_C_FLOAT SQL_C_DOUBLE	Data is within the range of the data type to which the number is being converted <sup>b</sup>	Data	Size of the C data type	N/A
	Data is outside the range of the data type to which the number is being converted <sup>b</sup>	Untouched	Untouched	22003
	Data is not a <i>numeric-literal</i> <sup>b</sup>	Untouched	Untouched	22005
SQL_C_BIT	Data is 0 or 1	Data	1 <sup>c</sup>	N/A
	Data is greater than 0, less than 2, and not equal to 1 <sup>a</sup>	Truncated data	1 <sup>c</sup>	01004
	Data is less than 0 or greater than or equal to 2 <sup>a</sup>	Untouched	Untouched	22003
	Data is not a <i>numeric-literal</i> <sup>a</sup>	Untouched	Untouched	22005
SQL_C_BINARY	Length of data ≤ <i>cbValueMax</i>	Data	Length of data	N/A
	Length of data > <i>cbValueMax</i>	Truncated data	Length of data	01004

(1 of 3)



<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_DATE	Data value is a valid <i>date-value</i> <sup>b</sup>	Data	6 <sup>c</sup>	N/A
	Data value is a valid <i>timestamp-value</i> ; time portion is zero <sup>b</sup>	Data	6 <sup>c</sup>	N/A
	Data value is a valid <i>timestamp-value</i> ; time portion is non-zero <sup>b, d</sup>	Truncated data	6 <sup>c</sup>	01004
	Data value is not a valid <i>date-value</i> or <i>timestamp-value</i> <sup>b</sup>	Untouched	Untouched	22008
SQL_C_TIME	Data value is a valid <i>time-value</i> <sup>t</sup>	Data	6 <sup>c</sup>	N/A
	Data value is a valid <i>timestamp-value</i> ; fractional seconds portion is zero <sup>b, e</sup>	Data	6 <sup>c</sup>	N/A
	Data value is a valid <i>timestamp-value</i> ; fractional seconds portion is non-zero <sup>b, e, f</sup>	Truncated data	6 <sup>c</sup>	01004
	Data value is not a valid <i>time-value</i> or <i>timestamp-value</i> <sup>b</sup>	Untouched	Untouched	22008
SQL_C_TIMESTAMP	Data value is a valid <i>timestamp-value</i> ; fractional seconds portion not truncated <sup>b</sup>	Data	16 <sup>c</sup>	N/A
	Data value is a valid <i>timestamp-value</i> ; fractional seconds portion truncated <sup>b</sup>	Truncated data	16 <sup>c</sup>	N/A
	Data value is a valid <i>date-value</i> <sup>b</sup>	Data <sup>g</sup>	16 <sup>c</sup>	N/A
	Data value is a valid <i>time-value</i> <sup>t</sup>	Data <sup>h</sup>	16 <sup>c</sup>	N/A
	Data value is not a valid <i>date-value</i> , <i>time-value</i> , or <i>timestamp-value</i> <sup>b</sup>	Untouched	Untouched	22008

<sup>b</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.

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
<sup>c</sup>	This is the size of the corresponding C data type.			
<sup>d</sup>	The time portion of <i>timestamp-value</i> is truncated.			
<sup>e</sup>	The date portion of <i>timestamp-value</i> is ignored.			
<sup>f</sup>	The fractional seconds portion of the time stamp is truncated.			
<sup>g</sup>	The time fields of the time-stamp structure are set to zero.			
<sup>h</sup>	The date fields of the time-stamp structure are set to the current date.			

(3 of 3)

When character SQL data is converted to numeric, date, time, or time-stamp C data, leading and trailing spaces are ignored.

All drivers that support date, time, and time-stamp data can convert character SQL data to date, time, or time-stamp C data, as specified in the previous table. Drivers might be able to convert character SQL data from other, driver-specific formats to date, time, or time-stamp C data. Such conversions are not interoperable among data sources.

## SQL to C: Numeric

The numeric ODBC SQL data types are found in the following list:

- SQL\_DECIMAL
- SQL\_REAL
- SQL\_SMALLINT
- SQL\_DOUBLE
- SQL\_INTEGER

The following table shows the ODBC C data types to which numeric SQL data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from SQL to C Data Types” on page C-13](#).

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_CHAR	Display size < <i>cbValueMax</i>	Data	Length of data	N/A
	Number of whole (as opposed to fractional) digits < <i>cbValueMax</i>	Truncated data	Length of data	01004
	Number of whole (as opposed to fractional) digits $\geq$ <i>cbValueMax</i>	Untouched	Untouched	22003
SQL_C_TINYINT SQL_C_UTINYINT SQL_C_TINYINT <sup>a</sup>	Data converted without truncation <sup>b</sup>	Data	Size of the C data type	N/A
SQL_C_SSHORT SQL_C_USHORT SQL_C_SHORT <sup>a</sup>	Data converted with truncation of fractional digits <sup>b</sup>	Truncated data	Size of the C data type	01004
SQL_C_SLONG SQL_C_ULONG SQL_C_LONG <sup>a</sup>	Conversion of data would result in loss of whole (as opposed to fractional) digits <sup>b</sup>	Untouched	Untouched	22003
SQL_C_FLOAT SQL_C_DOUBLE	Data is within the range of the data type to which the number is being converted <sup>b</sup>	Data	Size of the C data type	N/A
	Data is outside the range of the data type to which the number is being converted <sup>b</sup>	Untouched	Untouched	22003
SQL_C_BIT	Data is 0 or 1 <sup>b</sup>	Data	1 <sup>c</sup>	N/A
	Data is greater than 0, less than 2, and not equal to 1 <sup>b</sup>	Truncated data	1 <sup>c</sup>	01004
	Data is less than 0 or greater than or equal to 2 <sup>b</sup>	Untouched	Untouched	22003

(1 of 2)

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_BINARY	Length of data $\leq$ <i>cbValueMax</i>	Data	Length of data	N/A
	Length of data $>$ <i>cbValueMax</i>	Untouched	Untouched	22003

<sup>b</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>c</sup> This is the size of the corresponding C data type.

(2 of 2)

## SQL to C: Binary

The binary ODBC SQL data type that INFORMIX-CLI supports is SQL\_LONGVARBINARY.

The following table shows the ODBC C data types to which binary SQL data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from SQL to C Data Types” on page C-13](#).

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_CHAR	(Length of data) * 2 $<$ <i>cbValueMax</i>	Data	Length of data	N/A
	(Length of data) * 2 $\geq$ <i>cbValueMax</i>	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data $\leq$ <i>cbValueMax</i>	Data	Length of data	N/A
	Length of data $>$ <i>cbValueMax</i>	Truncated data	Length of data	01004

When binary SQL data is converted to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, a binary 00000001 is converted to “01” and a binary 11111111 is converted to “FF.”

The driver always converts individual bytes to pairs of hexadecimal digits and terminates the character string with a null byte. Because of this conversion, if *cbValueMax* is even and is less than the length of the converted data, the last byte of the *rgbValue* buffer is not used. (The converted data requires an even number of bytes, the next-to-last byte is a null byte, and the last byte cannot be used.)

## SQL to C: Date

The date ODBC SQL data type is SQL\_DATE.

The following table shows the ODBC C data types to which date SQL data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from SQL to C Data Types”](#) on page C-13.

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_CHAR	<i>cbValueMax</i> ≥ 11	Data	10	N/A
	<i>cbValueMax</i> < 11	Untouched	Untouched	22003
SQL_C_BINARY	Length of data ≤ <i>cbValueMax</i>	Data	Length of data	N/A
	Length of data > <i>cbValueMax</i>	Untouched	Untouched	22003
SQL_C_DATE	None <sup>a</sup>	Data	6 <sup>c</sup>	N/A
SQL_C_TIMESTAMP	None <sup>a</sup>	Data <sup>b</sup>	16 <sup>c</sup>	N/A

<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> The time fields of the time-stamp structure are set to zero.

<sup>c</sup> This is the size of the corresponding C data type.

When date SQL data is converted to character C data, the resulting string is in the “yyyy-mm-dd” format.

## SQL to C: Time Stamp

The time-stamp ODBC SQL data type is SQL\_TIMESTAMP.

The following table shows the ODBC C data types to which time-stamp SQL data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from SQL to C Data Types” on page C-13](#).

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
SQL_C_CHAR	<i>cbValueMax</i> > Display size	Data	Length of data	N/A
	$20 \leq cbValueMax \leq$ Display size	Truncated data <sup>b</sup>	Length of data	01004
	<i>cbValueMax</i> < 20	Untouched	Untouched	22003
SQL_C_BINARY	Length of data $\leq cbValueMax$	Data	Length of data	N/A
	Length of data > <i>cbValueMax</i>	Untouched	Untouched	22003
SQL_C_DATE	Time portion of time stamp is zero <sup>a</sup>	Data	6 <sup>f</sup>	N/A
	Time portion of time stamp is nonzero <sup>a</sup>	Truncated data <sup>c</sup>	6 <sup>f</sup>	01004
SQL_C_TIME	Fractional seconds portion of time stamp is zero <sup>a</sup>	Data <sup>d</sup>	6 <sup>f</sup>	N/A
	Fractional seconds portion of time stamp is nonzero <sup>a</sup>	Truncated data <sup>d,e</sup>	6 <sup>f</sup>	01004
SQL_C_TIMESTAMP	Fractional seconds portion of time stamp is not truncated <sup>a</sup>	Data <sup>e</sup>	16 <sup>f</sup>	N/A
	Fractional seconds portion of time stamp is truncated <sup>a</sup>	Truncated data <sup>e</sup>	16 <sup>f</sup>	01004

<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> The fractional seconds of the time stamp are truncated.

<sup>c</sup> The time portion of the time stamp is truncated.

<i>fCType</i>	Test	<i>rgbValue</i>	<i>pcbValue</i>	SQLSTATE
---------------	------	-----------------	-----------------	----------

<sup>d</sup> The date portion of the time stamp is ignored.

<sup>e</sup> The fractional seconds portion of the time stamp is truncated.

<sup>f</sup> This is the size of the corresponding C data type.

(2 of 2)

When time-stamp SQL data is converted to character C data, the resulting string is in the “yyyy-mm-dd hh:mm:ss[.f...]” format, where up to nine digits can be used for fractional seconds. Except for the decimal point and fractional seconds, the entire format must be used, regardless of the precision of the time-stamp SQL data type.

### SQL to C Data Conversion Examples

The following table illustrates how the driver converts SQL data to C data.

SQL Data Type	SQL Data Value	C Data Type	<i>cbValueMax</i>	<i>rgbValue</i>	SQLSTATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef\0 <sup>a</sup>	N/A
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde\0 <sup>a</sup>	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56\0 <sup>a</sup>	N/A
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234\0 <sup>a</sup>	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	----	22003
SQL_DECIMAL	1234.56	SQL_C_FLOAT	ignored	1234.56	N/A
SQL_DECIMAL	1234.56	SQL_C_SSHORT	ignored	1234	01004
SQL_DECIMAL	1234.56	SQL_C_STINYINT	ignored	----	22003
SQL_DOUBLE	1.2345678	SQL_C_DOUBLE	ignored	1.2345678	N/A
SQL_DOUBLE	1.2345678	SQL_C_FLOAT	ignored	1.234567	N/A
SQL_DOUBLE	1.2345678	SQL_C_STINYINT	ignored	1	N/A

(1 of 2)

## Converting Data from C to SQL Data Types

SQL Data Type	SQL Data Value	C Data Type	cbValueMax	rgbValue	SQLSTATE
SQL_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31\0 <sup>a</sup>	N/A
SQL_DATE	1992-12-31	SQL_C_CHAR	10	-----	22003
SQL_DATE	1992-12-31	SQL_C_TIMESTAMP	ignored	1992,12,31, 0,0,0,0 <sup>b</sup>	N/A
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12\0 <sup>a</sup>	N/A
SQL_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_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	----	22003

<sup>a</sup> “\0” represents a null-termination byte. The driver always null-terminates SQL\_C\_CHAR data.

<sup>b</sup> The numbers in this list are the numbers stored in the fields of the `TIMESTAMP_STRUCT` structure.

(2 of 2)

---

## Converting Data from C to SQL Data Types

When an application calls `SQLExecute` or `SQLExecDirect`, the driver retrieves the data for any parameters that are bound with `SQLBindParameter` from storage locations in the application. For data-at-execution parameters, the application sends the parameter data with `SQLPutData`. If necessary, the driver converts the data from the data type specified by the `fCType` argument in `SQLBindParameter` to the data type specified by the `fSqlType` argument in `SQLBindParameter`. Finally, the driver sends the data to the data source.

The word *convert* is used in this section in a broad sense, and it includes the transfer of data, without a conversion in data type, from one storage location to another.



The following table shows the supported conversions from ODBC C data types to ODBC SQL data types. A solid circle (●) indicates the default conversion for an SQL data type (the C data type from which the data is converted when the value of *fCType* is SQL\_C\_DEFAULT). A hollow circle (○) indicates a supported conversion.

C Data Type	SQL Data Type												
	SQL_CHAR	SQL_VARCHAR	SQL_LONGVARCHAR	SQL_DECIMAL	SQL_SMALLINT (signed)	SQL_SMALLINT (unsigned)	SQL_INTEGER (signed)	SQL_INTEGER (unsigned)	SQL_REAL	SQL_DOUBLE	SQL_LONGVARIABLE	SQL_DATE	SQL_TIMESTAMP
SQL_C_CHAR	●	●	●	●	○	○	○	○	○	○	○	○	○
SQL_C_BIT	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_STINYINT	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_UTINYINT	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_TINYINT	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_SSHORT	○	○	○	○	●	○	○	○	○	○	○	○	○
SQL_C_USHORT	○	○	○	○	○	●	○	○	○	○	○	○	○
SQL_C_SHORT	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_SLONG	○	○	○	○	○	○	●	○	○	○	○	○	○
SQL_C_ULONG	○	○	○	○	○	○	○	●	○	○	○	○	○
SQL_C_LONG	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_FLOAT	○	○	○	○	○	○	○	○	●	○	○	○	○
SQL_C_DOUBLE	○	○	○	○	○	○	○	○	○	●	○	○	○
SQL_C_BINARY	○	○	○	○	○	○	○	○	○	○	●	○	○
SQL_C_DATE	○	○	○	○	○	○	○	○	○	○	○	●	○
SQL_C_TIME	○	○	○	○	○	○	○	○	○	○	○	○	○
SQL_C_TIMESTAMP	○	○	○	○	○	○	○	○	○	○	○	○	●

The tables in the following sections describe how the driver or data source converts data sent to the data source; drivers are required to support conversions from all ODBC C data types to the ODBC SQL data types that they support. For a given ODBC C data type, the first column of the table lists the legal input values of the *fSqlType* argument in **SQLBindParameter**. The second column lists the outcomes of a test that the driver performs to determine if it can convert the data. The third column lists the SQLSTATE that is returned for each outcome by **SQLExecDirect**, **SQLExecute**, or **SQLPutData**. Data is sent to the data source only if SQL\_SUCCESS is returned.

If the *fSqlType* argument in **SQLBindParameter** contains a value for an ODBC SQL data type that is not shown in the table for a given C data type, then **SQLBindParameter** returns SQLSTATE 07006 (Restricted data type attribute violation). If the *fSqlType* argument contains a driver-specific value and the driver does not support the conversion from the specific ODBC C data type to the driver-specific SQL data type, then **SQLBindParameter** returns SQLSTATE S1C00 (Driver not capable).

If the *rgbValue* and *pcbValue* arguments specified in **SQLBindParameter** are both null pointers, then that function returns SQLSTATE S1009 (Invalid argument value). Although it is not shown in the tables, an application sets the value pointed to by the *pcbValue* argument of **SQLBindParameter** or the value of the *cbValue* argument to SQL\_NULL\_DATA to specify a NULL SQL data value. The application sets these values to SQL\_NTS to specify that the value in *rgbValue* is a null-terminated string.

The following terms are used in the tables:

- *Length of data* is the number of bytes of SQL data that are available to send to the data source, regardless of whether the data is truncated before it goes to the data source. For string data, this does not include the null-termination byte.
- *Column length* and *display size* are defined for each SQL data type in [“Precision, Scale, Length, and Display Size” on page C-8](#).
- *Number of digits* is the number of characters that represent a number, including the minus sign, decimal point, and exponent (if needed).
- Words in *italics* represent elements of the ODBC SQL grammar. For the syntax of grammar elements, see [Appendix B](#).

## C to SQL: Character

The character ODBC C data type is SQL\_C\_CHAR.

The following table shows the ODBC SQL data types to which C character data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from C to SQL Data Types”](#) on page C-24.

fSqlType	Test	SQLSTATE
SQL_CHAR	Length of data $\leq$ Column length	N/A
SQL_VARCHAR		
SQL_LONGVARCHAR	Length of data $>$ Column length	01004
SQL_DECIMAL	Data converted without truncation	N/A
SQL_SMALLINT		
SQL_INTEGER	Data converted with truncation of fractional digits.	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	22003
	Data value is not a <i>numeric-literal</i> .	22005
SQL_REAL	Data is within the range of the data type to which the number is being converted.	N/A
SQL_DOUBLE		
	Data is outside the range of the data type to which the number is being converted.	22003
	Data value is not a <i>numeric-literal</i> .	22005
SQL_LONGVARBINARY	(Length of data) / 2 $\leq$ Column length	N/A
	(Length of data) / 2 $>$ Column length	01004
	Data value is not a hexadecimal value.	22005

(1 of 2)

fSqlType	Test	SQLSTATE
SQL_DATE	Data value is a valid <i>ODBC-date-literal</i> .	N/A
	Data value is a valid <i>ODBC-timestamp-literal</i> ; time portion is zero.	N/A
	Data value is a valid <i>ODBC-timestamp-literal</i> ; time portion is non-zero. <sup>a</sup>	01004
	Data value is not a valid <i>ODBC-date-literal</i> or <i>ODBC-timestamp-literal</i> .	22008
SQL_TIMESTAMP	Data value is a valid <i>ODBC-timestamp-literal</i> ; fractional seconds portion not truncated.	N/A
	Data value is a valid <i>ODBC-timestamp-literal</i> ; fractional seconds portion truncated.	01004
	Data value is a valid <i>ODBC-date-literal</i> . <sup>d</sup>	N/A
	Data value is a valid <i>ODBC-time-literal</i> . <sup>e</sup>	N/A
	Data value is not a valid <i>ODBC-date-literal</i> , <i>ODBC-time-literal</i> , or <i>ODBC-timestamp-literal</i> .	22008

<sup>a</sup> The time portion of the time stamp is truncated.

<sup>b</sup> The date portion of the time stamp is ignored.

<sup>c</sup> The fractional seconds portion of the time stamp is truncated.

<sup>d</sup> The time portion of the time stamp is set to zero.

<sup>e</sup> The date portion of the time stamp is set to the current date.

(2 of 2)

When character C data is converted to date or time-stamp SQL data, leading and trailing blanks are ignored.

When character C data is converted to binary SQL data, each 2 bytes of character data are converted to a single byte (8 bits) of binary data. Each 2 bytes of character data represent a number in hexadecimal form. For example, “01” is converted to a binary 00000001 and “FF” is converted to a binary 11111111.

The driver always converts pairs of hexadecimal digits to individual bytes and ignores the null-termination byte. Because of this conversion, if the length of the character string is odd, the last byte of the string (excluding the null termination byte, if any) is not converted.

INFORMIX-CLI can convert character C data to date or time-stamp SQL data as specified in the previous table.

## **C to SQL: Numeric**

The numeric ODBC C data types are found in the following list:

- SQL\_C\_DOUBLE
- SQL\_C\_FLOAT
- SQL\_C\_LONG
- SQL\_C\_TINYINT
- SQL\_C\_SLONG
- SQL\_C\_SHORT
- SQL\_C\_SSHORT
- SQL\_C\_TINYINT
- SQL\_C\_UTINYIN
- SQL\_C\_ULONG
- SQL\_C\_USHORT

The following table shows the ODBC SQL data types to which numeric C data can be converted. For an explanation of the columns and terms in the table, see “[Converting Data from C to SQL Data Types](#)” on page C-24.

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Number of digits $\leq$ Column length	N/A
	Number of whole (as opposed to fractional) digits $\leq$ Column length	01004
	Number of whole (as opposed to fractional) digits $>$ Column length	22003
SQL_DECIMAL SQL_SMALLINT SQL_INTEGER	Data converted without truncation	N/A
	Data converted with truncation of fractional digits	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits.	22003
SQL_REAL SQL_DOUBLE	Data is within the range of the data type to which the number is being converted.	N/A
	Data is outside the range of the data type to which the number is being converted.	22003

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the numeric C data types. The driver assumes that the size of *rgbValue* is the size of the numeric C data type.

## C to SQL: Bit

The bit ODBC C data type is `SQL_C_BIT`.

The following table shows the ODBC SQL data types to which bit C data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from C to SQL Data Types” on page C-24](#).

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	None	N/A
SQL_DECIMAL SQL_SMALLINT SQL_INTEGER SQL_REAL SQL_DOUBLE	None	N/A

The value pointed to by the *pcbValue* argument of `SQLBindParameter` and the value of the *cbValue* argument of `SQLPutData` are ignored when data is converted from the bit C data type. The driver assumes that the size of *rgbValue* is the size of the bit C data type.

## C to SQL: Binary

The binary ODBC C data type is `SQL_C_BINARY`.

The following table shows the ODBC SQL data types to which binary C data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from C to SQL Data Types” on page C-24](#).

fSqlType	Test	SQLSTATE
SQL_CHAR	Length of data ≤ Column length	N/A
SQL_VARCHAR	Length of data > Column length	01004
SQL_LONGVARCHAR		
SQL_DECIMAL	Length of data = SQL data length <sup>a</sup>	N/A
SQL_SMALLINT	Length of data ≠ SQL data length <sup>a</sup>	22003
SQL_INTEGER		
SQL_REAL		
SQL_DOUBLE		
SQL_LONGVARBINARY	Length of data ≤ Column length	N/A
	Length of data > Column length	01004
SQL_DATE	Length of data = SQL data length	N/A
SQL_TIMESTAMP	<sup>a</sup> Length of data ≠ SQL data length <sup>a</sup>	22003

<sup>a</sup> The SQL data length is the number of bytes needed to store the data on the data source. This might be different than the column length, as defined “[Precision, Scale, Length, and Display Size](#)” on page C-8.

## C to SQL: Date

The date ODBC C data type is SQL\_C\_DATE.

The following table shows the ODBC SQL data types to which date C data can be converted. For an explanation of the columns and terms in the table, see “[Converting Data from C to SQL Data Types](#)” on page C-24.

fSqlType	Test	SQLSTATE
SQL_CHAR	Column length ≥ 10	N/A
SQL_VARCHAR	Column length < 10	22003
SQL_LONGVARCHAR	Data value is not a valid date.	22008

(1 of 2)



fSqlType	Test	SQLSTATE
SQL_DATE	Data value is a valid date.	N/A
	Data value is not a valid date.	22008
SQL_TIMESTAMP	Data value is a valid date.	N/A
	<sup>a</sup> Data value is not a valid date.	22008

<sup>a</sup> The time portion of the time stamp is set to zero.

(2 of 2)

For information about what values are valid in a SQL\_C\_DATE structure, see [“Extended C Data Types” on page C-5](#).

When date C data is converted to character SQL data, the resulting character data is in the “yyyy-mm-dd” format.

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the date C data type. The driver assumes that the size of *rgbValue* is the size of the date C data type.

## C to SQL: Time

The time ODBC C data type is SQL\_C\_TIME.

The following table shows the ODBC SQL data types to which time C data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from C to SQL Data Types” on page C-24.](#)

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Column length $\geq$ 8	N/A
	Column length < 8	22003
	Data value is not a valid time.	22008
SQL_TIMESTAMP	Data value is a valid time. <sup>a</sup>	N/A
	Data value is not a valid time.	22008

<sup>a</sup> The date portion of the time stamp is set to the current date, and the fractional seconds portion of the time stamp is set to zero.

For information about what values are valid in a SQL\_C\_TIME structure, see [“Extended C Data Types” on page C-5.](#)

When time C data is converted to character SQL data, the resulting character data is in the “hh:mm:ss” format.

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the time C data type. The driver assumes that the size of *rgbValue* is the size of the time C data type.

## C to SQL: Time Stamp

The time-stamp ODBC C data type is SQL\_C\_TIMESTAMP.

The following table shows the ODBC SQL data types to which time-stamp C data can be converted. For an explanation of the columns and terms in the table, see [“Converting Data from C to SQL Data Types” on page C-24.](#)

fSqlType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Column length $\geq$ Display size	N/A
	$19 \leq$ Column length $<$ Display size <sup>a</sup>	01004
	Column length $<$ 19	22003
	Data value is not a valid date.	22008
SQL_DATE	Time fields are zero.	N/A
	Time fields are non-zero. <sup>b</sup>	01004
	Data value does not contain a valid date.	22008
SQL_TIMESTAMP	Fractional seconds fields are not truncated.	N/A
	Fractional seconds fields are truncated. <sup>d</sup>	01004
	Data value is not a valid time stamp.	22008

<sup>a</sup> The fractional seconds of the time stamp are truncated.

<sup>b</sup> The time fields of the time-stamp structure are truncated.

<sup>c</sup> The date fields of the time-stamp structure are ignored.

<sup>d</sup> The fractional seconds fields of the time-stamp structure are truncated.

For information about what values are valid in a SQL\_C\_TIMESTAMP structure, see [“Extended C Data Types” on page C-5.](#)

When time-stamp C data is converted to character SQL data, the resulting character data is in the “yyyy-mm-dd hh:mm:ss[.f...]” format.

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the time-stamp C data type. The driver assumes that the size of *rgbValue* is the size of the time-stamp C data type.

## C to SQL Data Conversion Examples

The following table illustrates how the driver converts C data to SQL data.

C DataType	C Data Value	SQL Data Type	Column length	SQL Data Value	SQLSTATE
SQL_C_CHAR	abcdef\0 <sup>a</sup>	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef\0 <sup>a</sup>	SQL_CHAR	5	abcde	01004
SQL_C_CHAR	1234.56\0 <sup>a</sup>	SQL_DECIMAL	8 <sup>b</sup>	1234.56	N/A
SQL_C_CHAR	1234.56\0 <sup>a</sup>	SQL_DECIMAL	7 <sup>b</sup>	1234.5	01004
SQL_C_CHAR	1234.56\0 <sup>a</sup>	SQL_DECIMAL	4	----	22003
SQL_C_FLOAT	1234.56	SQL_INTEGER	not applicable	1234	01004
SQL_C_DATE	1992,12,31 <sup>c</sup>	SQL_CHAR	10	1992-12-31	N/A
SQL_C_DATE	1992,12,31 <sup>c</sup>	SQL_CHAR	9	----	22003
SQL_C_DATE	1992,12,31 <sup>c</sup>	SQL_TIMESTAMP	not applicable	1992-12-31 00:00:00.0	N/A
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000 <sup>d</sup>	SQL_CHAR	22	1992-12-31 23:45:55.12	N/A
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000 <sup>d</sup>	SQL_CHAR	21	1992-12-31 23:45:55.1	01004
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000 <sup>d</sup>	SQL_CHAR	18	----	22003

<sup>a</sup> “\0” represents a null-termination byte. The null-termination byte is required only if the length of the data is SQL\_NTS.

<sup>b</sup> An addition to bytes for numbers, one byte is required for a sign and another byte is required for the decimal point.

<sup>c</sup> The numbers in this list are the numbers stored in the fields of the DATE\_STRUCT structure.

<sup>d</sup> The numbers in this list are the numbers stored in the fields of the TIMESTAMP\_STRUCT structure.

---

# Comparison Between INFORMIX-CLI and Embedded SQL

This appendix compares INFORMIX-CLI and embedded SQL.

---

## INFORMIX-CLI to Embedded SQL

The following table compares INFORMIX-CLI functions (core ODBC) with embedded SQL statements. This comparison is based on the X/Open and SQL Access Group SQL CAE specification (1992).

INFORMIX-CLI uses a parameter marker in place of a host variable, where a host variable occurs in embedded SQL.

The SQL language is based on the X/Open and SQL Access Group SQL CAE specification (1992).

INFORMIX-CLI Function	Statement	Comments
<b>SQLAllocEnv</b>	none	Driver manager and driver memory allocation.
<b>SQLAllocConnect</b>	none	Driver manager and driver memory allocation.
<b>SQLConnect</b>	CONNECT	Association management.
<b>SQLAllocStmt</b>	none	Driver manager and driver memory allocation.
<b>SQLPrepare</b>	PREPARE	The prepared SQL string can contain any of the valid preparable functions as defined by the X/Open specification, including ALTER, CREATE, <i>cursor-specification</i> , searched DELETE, dynamic SQL positioned DELETE, DROP, GRANT, INSERT, REVOKE, searched UPDATE, or dynamic SQL positioned UPDATE.
<b>SQLBindParameter</b>	SET DESCRIPTOR	Dynamic SQL ALLOCATE DESCRIPTOR and dynamic SQL SET DESCRIPTOR. ALLOCATE DESCRIPTOR would normally be issued on the first call to <b>SQLBindParameter</b> for an <i>hstmt</i> . Alternatively, ALLOCATE DESCRIPTOR can be called during <b>SQLAllocStmt</b> , although this call would not be needed by SQL statements that do not contain embedded parameters. The descriptor name is generated by the driver.
<b>SQLSetCursorName</b>	none	The specified cursor name is used in the DECLARE CURSOR statement generated by <b>SQLExecute</b> or <b>SQLExecDirect</b> .
<b>SQLGetCursorName</b>	none	Driver cursor name management.

(1 of 3)

INFORMIX-CLI Function	Statement	Comments
<b>SQLExecute</b>	EXECUTE or DECLARE CURSOR and OPEN CURSOR	Dynamic SQL EXECUTE. If the SQL statement requires a cursor, then a dynamic SQL DECLARE CURSOR statement and a dynamic SQL OPEN are issued at this time.
<b>SQLExecDirect</b>	EXECUTE IMMEDIATE or DECLARE CURSOR and OPEN CURSOR	The INFORMIX-CLI function call provides for support for a <i>cursor specification</i> and statements allowed in an EXECUTE IMMEDIATE dynamic SQL statement. In the case of a <i>cursor specification</i> , the call corresponds to static SQL DECLARE CURSOR and OPEN statements.
<b>SQLNumResultCols</b>	GET DESCRIPTOR	COUNT form of dynamic SQL GET DESCRIPTOR.
<b>SQLColAttributes</b>	GET DESCRIPTOR	COUNT form of dynamic SQL GET DESCRIPTOR or VALUE form of dynamic SQL GET DESCRIPTOR with <i>field-name</i> in {NAME, TYPE, LENGTH, PRECISION, SCALE, NULLABLE}.
<b>SQLDescribeCol</b>	GET DESCRIPTOR	VALUE form of dynamic SQL GET DESCRIPTOR with <i>field-name</i> in {NAME, TYPE, LENGTH, PRECISION, SCALE, NULLABLE}.
<b>SQLBindCol</b>	none	This function establishes output buffers that correspond in usage to host variables for static SQL FETCH, and to an SQL DESCRIPTOR for dynamic SQL FETCH <i>cursor</i> USING SQL DESCRIPTOR <i>descriptor</i> .
<b>SQLFetch</b>	FETCH	Static or dynamic SQL FETCH. If the call is a dynamic SQL FETCH, then the VALUE form of GET DESCRIPTOR is used, with <i>field-name</i> in {DATA, INDICATOR}. DATA and INDICATOR values are placed in output buffers specified in <b>SQLBindCol</b> .

(2 of 3)

INFORMIX-CLI Function	Statement	Comments
<b>SQLRowCount</b>	GET DIAGNOSTICS	Requested field ROW_COUNT.
<b>SQLFreeStmt</b> (SQL_CLOSE option)	CLOSE	Dynamic SQL CLOSE.
<b>SQLFreeStmt</b> (SQL_DROP option)	none	Driver manager and driver memory deallocation.
<b>SQLTransact</b>	COMMIT WORK or COMMIT ROLLBACK	None.
<b>SQLDisconnect</b>	DISCONNECT	Association management.
<b>SQLFreeConnect</b>	none	Driver manager and driver memory deallocation.
<b>SQLFreeEnv</b>	none	Driver manager and driver memory deallocation.
<b>SQLCancel</b>	none	None.
<b>SQLError</b>	GET DIAGNOSTICS	GET DIAGNOSTICS retrieves information from the SQL diagnostics area that pertains to the most recently executed SQL statement. This information can be retrieved following execution and preceding the deallocation of the statement.

(3 of 3)



## Embedded SQL to INFORMIX-CLI

The following tables list the relationship between the X/Open embedded SQL language and corresponding INFORMIX-CLI functions. The section number that appears in the first column of each table refers to the section of the X/Open and SQL Access Group SQL CAE specification (1992).

### Declarative Statements

The following table lists declarative statements.

Section	SQL Statement	INFORMIX-CLI Function	Comments
4.3.1	Static SQL DECLARE CURSOR	none	Issued implicitly by the driver if a <i>cursor specification</i> is passed to <b>SQLExecDirect</b> .
4.3.2	Dynamic SQL DECLARE CURSOR	none	Cursor is generated automatically by the driver. To set a name for the cursor, use <b>SQLSetCursorName</b> . To retrieve a cursor name, use <b>SQLGetCursorName</b> .

### Data Definition Statements

The following table lists data definition statements.

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.1.2	ALTER TABLE	<b>SQLPrepare</b> , <b>SQLExecute</b> , or <b>SQLExecDirect</b>	None.
5.1.3	CREATE INDEX		
5.1.4	CREATE TABLE		
5.1.5	CREATE VIEW		
5.1.6	DROP INDEX		
5.1.7	DROP TABLE		
5.1.8	DROP VIEW		
5.1.9	GRANT		
5.1.9	REVOKE		

## Data Manipulation Statements

The following table lists data manipulation statements.

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.2.1	CLOSE	<b>SQLFreeStmt</b> (SQL_CLOSE option)	None.
5.2.2	Positioned DELETE	<b>SQLExecDirect</b> (..., "DELETE FROM <i>table-name</i> WHERE CURRENT OF <i>cursor-name</i> ")	Driver-generated <i>cursor-name</i> can be obtained by calling <b>SQLGetCursorName</b> .
5.2.3	Searched DELETE	<b>SQLExecDirect</b> (..., "DELETE FROM <i>table-name</i> WHERE <i>search-condition</i> ")	None.
5.2.4	FETCH	<b>SQLFetch</b>	None.
5.2.5	INSERT	<b>SQLExecDirect</b> (..., "INSERT INTO <i>table-name</i> ...")	Can also be invoked by <b>SQLPrepare</b> and <b>SQLExecute</b> .
5.2.6	OPEN	none	When a SELECT statement is specified, a cursor is opened implicitly by <b>SQLExecute</b> or <b>SQLExecDirect</b> .
5.2.7	SELECT ...INTO	none	Not supported.
5.2.8	Positioned UPDATE	<b>SQLExecDirect</b> (..., "UPDATE <i>table-name</i> SET <i>column-identifier</i> = <i>expression</i> ... WHERE CURRENT OF <i>cursor-name</i> ")	Driver-generated <i>cursor-name</i> can be obtained by calling <b>SQLGetCursorName</b> .
5.2.9	Searched UPDATE	<b>SQLExecDirect</b> (..., "UPDATE <i>table-name</i> SET <i>column-identifier</i> = <i>expression</i> ... WHERE <i>search-condition</i> ")	None.

## Dynamic SQL Statements

The following table lists dynamic SQL statements.

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.3 (see 5.2.1)	Dynamic SQL CLOSE	<b>SQLFreeStmt</b> (SQL_CLOSE option)	None.
5.3 (see 5.2.2)	Dynamic SQL Positioned DELETE	<b>SQLExecDirect</b> (..., "DELETE FROM <i>table-name</i> WHERE CURRENT OF <i>cursor-name</i> ")	Can also be invoked by <b>SQLPrepare</b> and <b>SQLExecute</b> .
5.3 (see 5.2.8)	Dynamic SQL Positioned UPDATE	<b>SQLExecDirect</b> (..., "UPDATE <i>table-name</i> SET <i>column-identifier</i> = <i>expression</i> ...WHERE CURRENT OF <i>cursor-name</i> ")	Can also be invoked by <b>SQLPrepare</b> and <b>SQLExecute</b> .
5.3.3	ALLOCATE DESCRIPTOR	None	Descriptor information is implicitly allocated and attached to the <i>hstmt</i> by the driver. Allocation occurs at either the first call to <b>SQLBindParameter</b> or at <b>SQLExecute</b> or <b>SQLExecDirect</b> time.
5.3.4	DEALLOCATE DESCRIPTOR	<b>SQLFreeStmt</b> (SQL_DROP option)	None.
5.3.5	DESCRIBE	none	None.
5.3.6	EXECUTE	<b>SQLExecute</b>	None.
5.3.7	EXECUTE IMMEDIATE	<b>SQLExecDirect</b>	None.
5.3.8	Dynamic SQL FETCH	<b>SQLFetch</b>	None.

(1 of 2)

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.3.9	GET DESCRIPTOR	<b>SQLNumResultCols</b> <b>SQLDescribeCol</b> <b>SQLColAttributes</b>	COUNT FORM. VALUE form with <i>field-name</i> in {NAME, TYPE, LENGTH, PRECISION, SCALE, NULLABLE}.
5.3.10	Dynamic SQL OPEN	<b>SQLExecute</b>	None.
5.3.11	PREPARE	<b>SQLPrepare</b>	None.
5.3.12	SET DESCRIPTOR	<b>SQLBindParameter</b>	<b>SQLBindParameter</b> is associated with only one <i>hstmt</i> where a descriptor is applied to any number of statements with USING SQL DESCRIPTOR.

(2 of 2)

## Transaction Control Statements

The following table lists transaction control statements.

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.4.1	COMMIT WORK	<b>SQLTransact</b> (SQL_COMMIT option)	None.
5.4.2	ROLLBACK WORK	<b>SQLTransact</b> (SQL_ROLLBACK option)	None.

## Association Management Statements

The following table lists association management statements.

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.5.1	CONNECT	<b>SQLConnect</b>	None.
5.5.2	DISCONNECT	<b>SQLDisconnect</b>	INFORMIX-CLI does not support DISCONNECT ALL.
5.5.3	SET CONNECTION	None	<p>The SQL Access Group (SAG) Call Level Interface allows multiple simultaneous connections to be established, but only one connection can be active at one time. SAG-compliant drivers track which connection is active and automatically switch to a different connection if a different connection handle is specified. However, the active connection must be in a state that allows the connection context to be switched; in other words, there must not be a transaction in progress on the current connection.</p> <p>Drivers that are not SAG-compliant are not required to support this behavior. That is, drivers that are not SAG compliant are not required to return an error if the driver, and its associated data source can simultaneously support multiple active connections.</p>

## Diagnostic Statement

The following table lists the GET DIAGNOSTIC statement.

Section	SQL Statement	INFORMIX-CLI Function	Comments
5.6.1	GET DIAGNOSTICS	<b>SQLException</b> <b>SQLRowCount</b>	For <b>SQLException</b> , the following fields from the diagnostics area are available:  RETURNED_SQLSTATE, MESSAGE_TEXT, and MESSAGE_LENGTH.  For <b>SQLRowCount</b> , the ROW_COUNT field is available.

# Index

## A

Access mode 13-105  
 Access plans 6-6  
 ALIAS table type 13-325  
 Aliases, column 13-193  
 ALTER TABLE statements  
   data type names B-1  
   interoperability 6-3  
   modifying syntax 6-3  
   qualifier usage in 13-207  
   supported clauses 13-193  
 API conformance  
   definition of 1-12  
 Application  
   interactive example 10-7  
   static example 10-4  
 Architecture  
   INFORMIX-CLI 1-3  
   ODBC error handling 8-5  
 Arguments  
   naming conventions 13-6  
   null pointers 3-6  
   pointers 3-5  
   search patterns 13-10  
   SQLTables 13-324  
   *See also* Parameters.  
 Arrays 11  
   *See* Binding columns, column-wise.  
   *See* Parameters, arrays.  
 Association management  
   statements D-9  
 Asterisk (\*) 13-46  
 Attributes  
   columns 13-57, 13-91  
   connection string

  for driver (UNIX) 2-13, 2-28  
   data source specification  
   for driver (UNIX) 2-9  
   definition of 1-9  
 Auto-commit mode  
   described 6-9  
   specifying 13-278  
   SQLTransact 13-329  
   *See also* Transactions.

## B

Batch statements  
   processing 13-229  
   syntax B-11  
 Binary data  
   converting to C C-20  
   converting to SQL C-31  
   literal length 13-201  
   null-termination byte 13-35  
   retrieving in parts 13-167, 13-174  
   specifying conversions  
     SQLBindCol 13-19  
     SQLBindParameter 13-31  
     SQLGetData 13-167  
   transferring C-8  
 Binary large object (blob). *See*  
   SQLGetData.  
 Binding columns  
   binding type 13-291  
   code examples 13-142  
   column attributes 7-4  
   column-wise 7-8, 13-134  
   null pointers 13-23  
   row-wise 7-8, 13-135  
   single row 7-4

- SQLBindCol 13-22
- unbinding 13-23, 13-156
- See also* Converting data.
- See also* Rowsets.
- Binding parameters. *See* Parameters, binding.
- Bit data
  - converting to SQL C-31
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
- Blob, binary large object. *See* SQLGetData.
- Block cursors 7-9
- Bookmarks
  - SQLExtendedFetch 13-140
- Boundaries, segment 3-5
- Braces ({} ) 13-46, 13-101
- Browse request connection
  - string 13-45
- Browse result connection
  - string 13-45
- Buffers
  - allocating 3-5
  - input 3-5
  - interoperability 3-5
  - maintaining pointers 3-5
  - NULL data 3-6
  - null pointers 3-6
  - null-termination 3-6
  - output 3-6
  - segment boundaries 3-5
  - truncating data 3-7
  - See also* NULL data.
  - See also* Null-termination byte.
- Bulk operations 13-241

## C

- C data types
  - conversion examples C-23, C-36
  - converting from SQL data
    - types C-13
  - converting to SQL data
    - types C-24
  - defaults C-6
  - defined C-4
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
  - using 3-8
  - See also* SQL data types.
- Call Level Interface (CLI)
  - support of 13-205
- Canceling
  - connection browsing 13-94
  - data-at-execution 13-53
- Cardinality 13-313
- Case-sensitivity
  - columns 13-58
  - data types 13-223
  - quoted SQL identifiers 13-208
- Catalog functions
  - list of 6-9
  - search patterns 13-10
  - summary 12-8
- Character data
  - case-sensitivity 13-223
  - converting to C C-15
  - converting to SQL C-27
  - empty string 3-5
  - literal
    - length 13-201
    - prefix string 13-222
    - suffix string 13-222
  - null-termination byte 3-6, 13-35
  - retrieving in parts 13-167, 13-174
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
- Character sets 5-9
- Characters, escape 13-10
- Characters, special. *See* Special characters.
- CLI. *See* Call Level Interface.
- Closing cursors 13-156, 13-194
  - SQLCancel 13-52
  - SQLFreeStmt 9-3
- Clustered indexes 13-312
- Code examples
  - ad hoc query 10-7
  - parameter values 6-8
  - static SQL 10-4
- See also* specific function description.
- Codes, return 8-3
- Collating 13-203, 13-312
- Columns
  - aliases 13-193
  - attributes 13-57, 13-91
  - binding. *See* Binding columns.
  - in GROUP BY clauses 13-201
  - in indexes 13-201
  - in select list 13-202
  - in tables 13-202
  - listing 13-71
  - maximum name length 13-201
  - nullability 13-203
  - number of 13-58, 13-235
  - precision. *See* Precision.
  - primary keys 13-255
  - pseudo 13-305
  - signed 13-61
  - unbinding 13-23, 13-156
  - uniquely identifying row 13-303
  - See also* Results sets.
  - See also* Retrieving data.
  - See also* SQL data types.
  - See also* Tables.
- Comment Icons Intro-9
- COMMIT statements
  - interoperability 6-9, 13-120
  - SQLExecute 13-127
  - SQLPrepare 13-250
- Committing transactions 13-329
- COMMIT, in
  - SQLExecDirect 13-120
- Compliance, icons Intro-10
- Concurrency
  - defined 7-12
  - serializability 7-12
  - specifying 13-287
  - supported types 13-208
- ConfigDSN
  - function description 14-3
  - with
    - SQLRemoveDSNFromIni 14-6
  - with SQLWriteDSNToIni 14-6
  - with
    - SQLWritePrivateProfileString 14-6



- ConfigTranslator
  - function description 14-7
  - See also Translation setup shared library.
- Configuring data sources. See Data sources, configuring.
- Conformance
  - API levels 1-12
  - ODBC standards 1-12
  - SQL grammar 1-14
- Conformance levels
  - by function 13-179
  - determining 13-205
  - in ODBC 1.0 7-14
  - SQL B-4, C-2
  - See also API conformance.
  - See also SQL conformance.
- Connection handles
  - active 13-192
  - allocating 5-5, 13-13
  - defined 3-7
  - freeing 9-4
  - overwriting 13-13
  - SQLFreeConnect 13-152
  - with threads 13-13
- Connection options
  - access mode 13-105
  - commit mode 13-278
  - dialog boxes 13-280
  - login interval 13-105
  - maximum length 13-162, 13-277
  - packet size 13-279
  - releasing 13-277
  - reserved 13-277
  - retrieving 13-160
  - setting 13-277
  - trace file 13-279
  - tracing 13-279
- Connection string
  - for Informix driver (UNIX) 2-13, 2-28
- Connection strings
  - browse request 13-45
  - browse result 13-45
  - special characters 13-45, 13-46
  - SQLDriverConnect 5-6, 13-101
- Connections
  - dialog boxes 13-102, 13-104, 13-105
  - disconnecting 9-4
  - driver manager 13-80
  - function summary 12-4, 12-9
  - in embedded SQL D-9
  - login information 5-8
  - SQLBrowseConnect 13-47
  - SQLConnect 13-80
  - SQLDisconnect 13-94
  - SQLDriverConnect 13-101
  - terminating browsing 13-48
- Converting data
  - C to SQL C-24
  - default conversions C-6
  - examples C-23, C-36
  - hexadecimal characters C-20, C-29, C-31
  - parameters 6-7
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
  - SQL to C C-13
  - SQLExtendedFetch 13-134
  - SQLFetch 13-22
  - SQLGetData 13-174
  - See also Translation shared libraries.
- Core API
  - for INFORMIX driver 1-13
  - list of functions 1-13
- CREATE TABLE statements
  - data type names B-1
  - interoperability 6-3
  - modifying syntax 6-3
  - NOT NULL clauses 13-203
- Currency columns 13-59
- Currency data types 13-224
- Cursor position
  - errors 13-139
  - required 7-13, 13-174
  - SQLExtendedFetch 7-12, 13-136, 13-140
  - SQLFetch 13-150
- Cursors
  - block 7-9
  - closing cursors
    - SQLCancel 13-52
    - SQLFreeStmt 9-3
  - described 7-6

- dynamic 7-10
- getting names 13-165
- holes in 7-11, 13-210
- keyset-driven 7-10
- maximum name length 13-202
- mixed 7-11
- positioned statements 7-14
- position. See Cursor position.
- scrollable 7-10
- sensitivity 13-210
- setting names 13-285
- simulated 13-303
- specifying type 7-12
- SQLFreeStmt 13-156
- static 7-10
- supported types 13-208
- transaction behavior 13-330
- See also Concurrency.
- See also Cursor library.

---

## D

- Data
  - converting. See Converting data.
  - long See Long data values.
  - retrieving. See Retrieving data.
  - transferring in binary form C-8
  - translating. See Translation shared libraries.
  - truncating. See Truncating data.
- Data Definition Language (DDL)
  - in embedded SQL D-5
- Data Manipulation Language (DML)
  - in embedded SQL D-6
  - qualifier usage in 13-207
- Data source name attribute
  - for driver (UNIX) 2-9
- Data source specification entry
  - for driver (UNIX) 2-9
- Data sources
  - adding 14-5
  - configuring 14-6
  - default 5-6
  - deleting 14-7
  - information types 13-189
  - listing 13-86
  - names 13-194

- read only 13-194
- UNIX
  - adding and modifying 2-6
  - connecting to 2-11
- Windows
  - adding and modifying 2-16
- Data translation. *See* Translation shared libraries.
- Data Types
  - supported by INFORMIX driver 1-15
- Data types
  - supported by INFORMIX-CLI driver 1-15
  - See* C data types.
  - See* SQL data types.
- Data-at-execution
  - canceling 13-53
  - macro 13-30, 13-33, 13-35, 13-36
  - parameters 6-10
  - SQLBindParameter 13-30, 13-33, 13-35, 13-36
- DATABASE attribute
  - for Informix driver (UNIX) 2-14, 2-29
- Database name attribute
  - for driver (UNIX) 2-9
- Databases
  - current 13-278
  - information types 13-188
- Date data
  - converting to C C-21
  - converting to SQL C-32
  - intervals 13-211
  - literals 6-13
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
- Date functions
  - supported by INFORMIX driver 1-15
- DBMS product information 13-188
- DDL. *See* Data Definition Language.
- Declarative statements D-5
- Default data source
  - specification 1-10
- DELETE statements
  - See also* SQLSetPos.
  - affected rows 13-272
  - qualifier usage in 13-207
- Deletes, positioned. *See* Positioned delete statements.
- Deleting cursors 13-194
- Delimiter character, SQL
  - identifiers 13-207
- Deprecated functions 13-287
- Descriptors
  - columns 13-57, 13-91
- Diagnostic statements D-10
- Dialog boxes
  - disabling 13-280
  - SQLDriverConnect 13-102, 13-104
- Dirty reads 13-196
- Display size 13-58, C-12
- DML. *See* Data Manipulation Language.
- Documentation notes Intro-14
- Driver keyword 13-101
  - version compatibility 13-47
- Driver manager
  - allocating handles 13-14
  - described 3-4
  - errors 8-5, 8-7
  - functions supported 13-179
  - listing drivers 13-109
  - loading drivers 13-80
  - ODBC version supported 13-205
  - SQLBrowseConnect 13-47
  - SQLConnect 13-80
  - SQLDriverConnect 13-102
  - SQLError 13-112
  - SQLSTATE values 13-9
  - tracing 13-155, 13-279
  - transactions 13-329
  - unloading drivers 13-81
- Driver path attribute
  - for driver (UNIX) 2-9
- Driver setup shared library
  - function summary 12-10
- Drivers
  - allocating handles 13-14
  - errors 8-3
  - file usage 13-198
  - functions supported 13-179
  - information types 13-188
  - Informix (UNIX) ?? to 1-16
  - INFORMIX-CLI (UNIX) 2-3
  - INFORMIX-CLI (Windows) 2-15
  - keywords 13-109
  - listing installed 13-109
  - loading 13-80
  - SQLError 13-112
  - SQLSTATE values 13-9
- DROP INDEX statements 13-312
- DSN attribute
  - for Informix driver (UNIX) 2-14, 2-29
- DSN keyword 13-101
- Dynamic cursors 7-10
- Dynamic SQL D-7

---

**E**

- Elements, SQL statements B-11
- Embedded SQL
  - executing statements 6-5
  - ODBC function equivalents D-1, D-5, D-6
- Empty strings 3-5, 13-11
- Environment handles
  - allocating 13-15
  - defined 3-7
  - freeing 9-4
  - initializing 5-4
  - overwriting 13-15
  - SQLFreeEnv 13-154
  - with threads 13-15
  - See also* Handles.
- Errors
  - application processing 8-8
  - clearing 13-112
  - format 8-5
  - messages 8-4
  - queues 13-112
  - return codes 8-3
  - rowsets 13-138
  - source 8-4
  - SQLError 13-111
  - SQLSTATE values A-1
  - with parameter arrays 13-244
  - See also* specific function description.
- Escape characters 13-10

Escape clauses  
  datetime literals 6-13  
  outer joins 6-16  
  procedures 6-17  
  scalar functions 6-14  
  support of 13-201  
  syntax 6-12

Examples  
  ad hoc query 10-7  
  data conversion C-23, C-36  
  parameter values 6-8  
  static SQL 10-4

Expressions  
  data types B-11  
  in ORDER BY clauses 13-197

Expressions in NewEra statements  
  expression types 6-13

Extensions to SQL 6-12

---

## F

Fat cursors 7-9

Fetching data. *See* Retrieving data.

Files  
  ODBC.INI 1-7, 1-10  
  odbc.ini file 1-6  
  sql.log 1-11  
  trace 1-10, 13-279  
  usage 13-198  
  .odbc.ini 2-6

Filtered indexes 13-313

Floating point data  
  converting to C C-18  
  converting to SQL C-29  
  specifying conversions  
    SQLBindCol 13-19  
    SQLBindParameter 13-31  
    SQLGetData 13-167

Freeing handles 13-152, 13-154,  
  13-159

Functionality, driver 13-179, 13-188

Functions, ODBC  
  buffers 3-5  
  canceling 13-52  
  deprecated 13-287  
  list of 12-3  
  return codes 8-3

SQL statement equivalents D-1,  
  D-5, D-6, D-7, D-8, D-9, D-10  
  supported 13-179  
  *See also* specific function.

---

## G

Global Language Support  
  (GLS) Intro-7, 2-4, 2-15

GLOBAL TEMPORARY table  
  type 13-325

Global transactions 13-329

Grammar, SQL B-11

Granting privileges 13-66, 13-318

GROUP BY clauses 13-200, 13-201

---

## H

.h files 3-3

Handles  
  connection. *See* Connection  
    handles.  
  defined 3-7  
  error queues 13-112  
  library 13-197  
  SQLAllocConnect 13-13  
  SQLAllocEnv 13-15  
  SQLAllocStmt 13-17  
  SQLFreeConnect 13-152  
  SQLFreeEnv 13-154  
  SQLFreeStmt 13-159  
  statement. *See* Statement handles.

Hashed indexes 13-312

Header files  
  required 3-3  
  sqlext.h  
    C data types 3-8  
    contents 13-9  
  sql.h  
    contents 13-9  
    SQL data type C-2

henv. *See* Environment handles.

Hexadecimal characters C-20,  
  C-29, C-31

Holes in cursors 7-11, 13-210

---

## I

Icons, compliance Intro-10

Identifiers, quoted  
  case-sensitive 13-208

IEF. *See* Integrity enhancement  
  facility.

Indexes  
  cardinality 13-313  
  clustered 13-312  
  collating 13-312  
  filtered 13-313  
  hashed indexes 13-312  
  listing 13-310  
  maximum columns in 13-201  
  maximum length 13-202  
  pages 13-313  
  qualifier usage in 13-207  
  sorting 13-312  
  unique 13-312  
  *See also* SQLStatistics.

Information, status  
  retrieving 8-4

INFORMIX 5 driver  
  isolation level 1-16  
  lock level 1-16

INFORMIX 7 driver  
  isolation level 1-16  
  lock level 1-16

INFORMIX driver  
  API conformance 1-12  
  data types 1-15  
  Isolation and lock levels 1-16  
  SQL conformance 1-14  
  supported ODBC extensions to  
    SQL 1-14

Informix driver (UNIX)  
  data source connection  
    using connection string 2-13,  
    2-28

Informix driver (Windows)  
  adding data sources 2-16

INFORMIX-CLI  
  architecture 1-3  
  Data types 1-15

INFORMIX-CLI driver  
  data types supported 1-15  
  for UNIX  
    adding data sources 2-6

- data source connection 2-11
- PATH variable setting 2-4
- setting INFORMIXDIR
  - environment variable 2-4
- for Windows
  - modifying data sources 2-24
  - system requirements 2-15
- INFORMIXDIR environment
  - variable 2-4
- Input buffers 3-5
- Input parameters 13-30
- Input/output parameters 13-31
- INSERT statements
  - affected rows 13-272
  - privileges 13-67
  - qualifier usage in 13-207
  - SQLParamOptions 13-241
  - See also* SQLSetPos.
- Installation. *See* odbcinstr.ini file.
- Integer data
  - converting to C C-18
  - converting to SQL C-29
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
- Integrity enhancement facility (IEF) 13-205, B-1
- Interoperability
  - affected rows 7-4
  - buffer length 3-5
  - cursor names 13-285
  - default C data type C-6
  - functionality 13-188
  - functions 13-188
  - procedure parameters 6-17
  - pseudo-columns 13-305
  - SQL statements
    - ALTER TABLE 6-3, 13-218
    - COMMIT 6-9, 13-120, 13-127, 13-250
    - CREATE TABLE 6-3, 13-218
    - ODBC extensions 6-12
    - ROLLBACK 13-120, 13-127, 13-250
    - syntax B-1
  - SQLGetData 7-7, 7-9, 13-174
  - transactions 6-9
  - transferring data C-8

- Intervals, datetime 13-211
- Isolation level
  - INFORMIX 5 driver 1-16
  - INFORMIX 7 driver 1-16
- Isolation levels, transaction 13-196

---

## J

- Joins, outer
  - described 6-16
  - support of 13-205

---

## K

- Keys
  - primary 13-255
- Keyset-driven cursors 7-10
- Keywords
  - data source-specific 13-201
  - in SQLBrowseConnect 13-45
  - in SQLDriverConnect 13-101
  - ODBC B-21

---

## L

- Length, available
  - SQLBindParameter 13-36
  - SQLExtendedFetch 13-135
  - SQLFetch 13-150
  - SQLGetData 13-173
- Length, buffer
  - input 3-5
  - output 3-6
  - SQLBindCol 13-23
  - SQLBindParameter 13-34
  - SQLGetData 13-173
- Length, column
  - defined C-11
  - result sets 13-87
  - tables 13-73, 13-305
- Length, data-at-execution 13-35, 13-36
- Length, maximum
  - buffers 3-5
  - column names length 13-201
  - columns 13-150, 13-173
  - connection options 13-162, 13-277

- cursor names 13-202, 13-285
- data 13-294
- error messages 13-111
- indexes 13-202
- owner name 13-202
- procedure names 13-202
- qualifiers 13-202
- rows 13-202
- statement options 13-217, 13-291
- user names 13-203
- Length, unknown
  - in length C-11
  - in precision C-9
  - SQLBindParameter 13-36
  - SQLExtendedFetch 13-135
  - SQLFetch 13-150
  - SQLGetData 13-173
- Level 1 API
  - for INFORMIX driver 1-13
  - list of functions 1-13
- Level 2 API
  - for INFORMIX driver 1-13
  - list of functions 1-13
- Levels, conformance. *See* Conformance levels.
- Library handles, driver 13-197
- LIKE predicates
  - support of 13-201
- Limitations, SQL statements 13-191
- Literals
  - character 13-201
  - date 6-13
  - prefix string 13-222
  - procedure parameters 6-17
  - suffix string 13-222
  - time 6-13
  - timestamp 6-13
- Loading drivers 13-80
- LOCAL TEMPORARY table
  - type 13-325
- Lock level
  - INFORMIX 5 driver 1-16
  - INFORMIX 7 driver 1-16
- Login authorization
  - connection strings
    - description 5-6
    - SQLBrowseConnect 13-45, 13-104
  - example 13-81

- interval period 13-105
- time-out 13-105
- See also* Connections.
- Long data values
  - length required 13-203
  - retrieving 7-7
  - sending
    - in parameters 6-10
- SQLBindParameter 13-36
- SQLGetData 13-174

---

## M

- Macros
  - data-at-execution 13-30, 13-33, 13-35, 13-36
- Manual-commit mode
  - beginning transactions 13-330
  - described 6-9
  - specifying 13-278
  - SQLTransact 13-330
  - See also* Transactions.
- Memory, allocating
  - buffers 3-5
  - connection handles 3-7
  - environment handles 3-7
  - result sets 7-4
  - rowsets 7-8
  - SQLAllocConnect 13-13
  - SQLAllocEnv 13-15
  - SQLAllocStmt 13-17
  - SQLFreeConnect 13-152
  - SQLFreeEnv 13-154
  - SQLFreeStmt 13-159
  - statement handles 3-7
- Messages, error. *See* Errors.
- Mixed cursors 7-11
- Modes
  - access 13-105
  - auto-commit. *See* Auto-commit mode.
  - manual-commit. *See* Manual-commit mode.
- Money columns 13-59
- Money data types 13-224
- Multiple-tier drivers
  - error messages 8-6
  - identifying data sources 8-6

- Multithreading
  - with connection handles 13-13
  - with environment handles 13-15
  - with statement handles 13-18

---

## N

- Names
  - arguments 13-6
  - cursors 13-165, 13-285
  - driver 13-197
  - index 13-312
  - localized data types 13-225
  - procedure 13-261
  - See also* Terms.
  - server 13-209
  - tables 13-324, 13-325
  - user 13-213
- Network traffic
  - maximum data length 13-294
  - packet size 13-279
  - prepared statements 6-6
- Nonrepeatable reads 13-196
- NOT NULL clauses 13-203
- NULL data
  - collating 13-203
  - concatenation behavior 13-193
  - output buffers 3-6
  - retrieving 13-23
  - SQLBindParameter 13-35
  - SQLExtendedFetch 13-135
  - SQLFetch 13-150
  - SQLGetData 13-173
  - SQLPutData 13-265
- Null pointers
  - input buffers 3-6
  - output buffers 3-6
  - parameter length 13-35
  - unbinding columns 13-23
- Nullability
  - columns 13-59, 13-74, 13-89
- Null-termination byte
  - binary data 13-35
  - character data 13-35
  - embedded 3-6
  - examples C-23, C-36
  - input buffers 3-5
  - output buffers 3-6

- parameters 6-8
- Numeric data
  - converting to C C-18
  - converting to SQL C-29
  - currency data type 13-224
  - money data type 13-224
  - radix 13-74
  - scalar functions 13-204
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
  - See also* Floating point data.
  - See also* Integer data.
- Numeric functions
  - supported by INFORMIX driver 1-14

---

## O

- ODBC
  - conformance standards 1-12
  - extensions to SQL
    - for INFORMIX driver 1-14
  - functions. *See* Functions, ODBC.
  - version compatibility. *See* Version compatibility.
- ODBC Administrator
  - adding data sources
    - for Informix driver 2-16
- ODBC administrator
  - modifying data sources
    - for driver (Windows) 2-24
- ODBC architecture
  - error handling 8-5
- ODBC options 1-10
- ODBC.INI
  - Windows 95 1-7, 2-17
  - Windows NT 1-7, 2-17
- odbc.ini
  - keywords
    - trace 13-15
  - listing drivers 13-109
- ODBC functions using
  - SQLAllocEnv 13-15
  - SQLBrowseConnect 13-47
  - SQLFreeEnv 13-155
  - SQLSetConnectOption 13-279

- Setup shared library functions
  - using ConfigDSN 14-6
- structure 1-6
- .odbc.ini (UNIX)
  - data source specification
    - section 1-9
  - default data source specification
    - section 1-10
  - file example 1-12
  - file format 1-8
  - for driver 2-10
  - modifying 2-6
  - ODBC data sources section 1-8
  - ODBC options section 1-10
- ODBC.INI (Windows)
  - data source specification
    - section 1-9
  - default data source specification
    - section 1-10
  - definition of 1-7, 1-10
  - file example 1-11
  - file format 1-8
  - ODBC data sources section 1-8
  - ODBC options section 1-10
- Optimistic concurrency control. *See*
  - Concurrency.
- ORDER BY clauses
  - columns in select list 13-205
  - expressions in 13-197
  - maximum columns in 13-201
- Outer joins
  - described 6-16
  - support of 13-205
- Output buffers 3-6
- Output parameters 13-31
- Owner names
  - procedure 13-261
  - table 13-318, 13-324
  - SQLColAttributes 13-59
  - SQLColumnPrivileges 13-66
  - SQLColumns 13-72
  - term, vendor-specific 13-206

---

## P

- Packet size 13-279
- Pages, index or table 13-313
- Parameters

- arrays 13-34, 13-241
- binding 6-8, 13-30
- data types B-2
- data-at-execution 6-10, 13-30,
  - 13-33, 13-35, 13-36
- default 13-31
- input 13-30
- input/output 13-31
- interoperability 6-17
- long data values 6-10
- markers 6-18, B-3
- number of 13-233
- output 13-31
- preparing 13-250
- procedure 6-18, 13-30
- return values 6-17
- unbinding 6-8, 13-156
- values 6-7
- version compatibility 13-287
- See also* Arguments.
- Patterns, search patterns 13-10
- Percent sign (%) 13-10, 13-324
- Phantoms 13-196
- Plans, access 6-6
- Platform icons Intro-10
- Pointers, maintaining 3-5
- Pointers, null. *See* Null pointers.
- Positioned DELETE statements
  - cursor name 13-165, 13-285
  - cursor position 13-136
  - executing 7-13
  - support of 13-206
  - version compatibility 7-14
- Positioned UPDATE statements
  - cursor name 13-165, 13-285
  - cursor position 13-136
  - executing 7-13
  - support of 13-206
  - version compatibility 7-14
- Position, cursor. *See* Cursor
  - position.
- Precision
  - columns
    - result sets 13-60, 13-89
    - tables 13-73, 13-304
  - data types 13-222
  - defined C-9
- Prepared statements,
  - deleting 13-194

- Preparing statements 6-6, 13-250
- Preserving cursors 13-194
- Primary keys 13-255
- Privileges
  - data source 13-194
  - grantable 13-67, 13-319
  - grantee 13-66, 13-318
  - grantor 13-66, 13-318
  - qualifier usage in 13-207
  - table user granting 13-66
- Procedure columns
  - parameters 13-30
- Procedures
  - defined 13-257
  - interoperability 6-17
  - name 13-261
  - name, maximum length 13-202
  - ODBC syntax 6-17
  - owner name 13-261
  - qualifier 13-261
  - qualifier usage in 13-207
  - return values 13-31
  - See also* Procedure
    - columns 13-261
    - support of 13-206
    - term, vendor-specific 13-206
- Pseudo-columns 13-305

---

## Q

- Qualifiers
  - current 13-278
  - index 13-312
  - maximum length 13-202
  - primary key 13-255
  - procedure 13-261
  - separator 13-207
  - table 13-60, 13-66, 13-72, 13-311,
    - 13-324
  - term, vendor-specific 13-207
  - usage 13-207
- Queries. *See* SQL statements.
- Question mark (?)
  - browse result connection
    - string 13-46
  - parameter markers 13-120, 13-250

Queues, error 13-112  
Quoted identifiers  
  case-sensitivity 13-208

---

## R

Radix 13-74  
Read committed isolation  
  level 13-196  
Read only  
  access mode 13-105  
  data sources 13-194  
Read uncommitted isolation  
  level 13-196  
Reads, dirty 13-196  
Reads, nonrepeatable 13-196  
Read/write access mode 13-105  
REFERENCES statements 13-67  
Referential integrity 13-205, B-1  
Refreshing data  
  SQLExtendedFetch 13-140  
Release notes Intro-14  
Remarks 13-74, 13-325  
Repeatable read isolation  
  level 13-196  
Result sets  
  arrays. *See* Rowsets.  
  binding columns 7-4  
  column attributes 7-4  
  described 4-3, 7-3  
  fetching data 7-5  
  fetching rowsets 7-9  
  multiple 13-203  
  number of columns 7-4  
  number of rows 7-4  
  retrieving data in parts 7-7  
  rowset size 7-7  
  SQLBindCol 13-22  
  SQLColAttributes 13-57  
  SQLDescribeCol 13-87  
  SQLFreeStmt 13-156  
  SQLMoreResults 13-229  
  SQLNumResultCols 13-235  
  SQLRowCount 13-272  
  *See also* Cursors.  
  *See also* Rows.  
Result states. *See* State transitions.  
Retrieving data

arrays. *See* Rowsets.  
binding columns. *See* Binding  
  columns.  
cursor position 13-136, 13-150  
disabling 13-295  
fetching data 7-5  
fetching rowsets 7-9  
in parts 7-7, 13-174  
long data values 7-7  
maximum length 13-294  
multiple result sets 13-203  
NULL data 13-23, 13-150, 13-173  
retrieving 13-295  
rows. *See* Rows.  
SQLExtendedFetch 13-134  
SQLFetch 13-150  
SQLGetData 13-174  
truncating data 13-150, 13-174  
unbound columns 7-7  
Return codes 8-3  
Return values, procedure 13-31  
ROLLBACK  
  SQLExecDirect 13-120  
ROLLBACK statements  
  cursor behavior 13-194  
  interoperability 6-9, 13-120,  
    13-127  
  SQLExecute 13-127  
  SQLPrepare 13-250  
Rolling back transactions 6-9,  
  13-329  
Row status array  
  errors 13-138  
  SQLExtendedFetch 13-141  
Row versioning isolation  
  level 13-196  
ROWID 13-303, 13-305  
Rows  
  affected 13-272  
  after last row 13-140  
  deleting 7-13  
  errors in 13-138  
  interoperability 7-4  
  maximum 13-294  
  maximum length 13-202  
  SQLExtendedFetch 13-141  
  updating 7-13  
  *See also* Cursors.  
  *See also* Retrieving data.

*See also* Rowsets.  
Rowsets  
  allocating 7-8  
  binding 7-7  
  binding type 13-291  
  cursor position 13-136  
  errors in 13-138  
  retrieving 7-9  
  size 13-295  
  SQLExtendedFetch 13-133  
  status 13-141

---

## S

SAG CLI compliance 13-205  
Scalar functions  
  information types 13-192  
  supported by INFORMIX-CLI  
    driver 1-14  
  syntax 6-14  
  TIMESTAMPADD  
    intervals 13-211  
  TIMESTAMPDIFF  
    intervals 13-211  
Scale  
  columns  
    result sets 13-89  
    tables 13-73, 13-305  
  data types 13-225  
  defined C-10  
Scope, rowid 13-306  
Scrollable cursors 7-10  
Search patterns 13-10  
Searchability  
  columns 13-60  
  data types 13-224  
Segment boundaries 3-5  
SELECT FOR UPDATE  
  statements 7-14, 13-206  
Select list  
  maximum columns in 13-202  
  ORDER BY columns in 13-205  
SELECT statements  
  affected rows 13-272  
  bulk 13-241  
  cursor name 13-162, 13-285  
  maximum tables in 13-203  
  multiple result sets 7-13, 13-229

- privileges 13-67
- qualifier usage in 13-207
- reexecuting 13-127
- UNION clauses 13-213
- See also* Result sets.
- Sensitivity, cursor 13-210
- Separator, qualifier 13-207
- Serializability 7-12
- Serializable isolation level 13-196
- Server name 13-209
- Shared library. *See* specific shared library.
- Signed columns 13-61
- Signed data types 13-224
- Simulated cursors
  - by applications 13-303
- Size, display 13-58, C-12
- Sorting 13-203, 13-313
- Special characters
  - in search patterns 13-10
  - in SQL identifiers 13-209
  - in SQLBrowseConnect 13-45, 13-46
- SQL
  - conformance
    - definition of 1-14
  - data types. *See* SQL data types.
  - dynamic D-7
  - embedded
    - executing statements 6-5
    - ODBC function
      - equivalents D-1, D-5, D-6
  - information types 13-190, 13-191
  - interoperability 6-3, B-1
  - keywords B-21
  - ODBC extensions to 6-12
  - ODBC grammar B-1
  - referential integrity B-1
  - static 10-4
  - See also* SQL statements.
- SQL Access Group 13-205
- SQL data types
  - columns
    - indexes 13-312
    - result sets 13-57
    - tables 13-73, 13-222
  - conformance levels C-2
  - conversion examples C-23, C-36
  - converting from C data types
    - types C-24
  - converting to C data types C-13
  - default C data types C-6
  - display size C-12
  - in translation shared
    - libraries 15-4, 15-7
  - length C-11
  - parameters B-2
  - precision C-9
  - scale C-10
  - specifying conversions
    - SQLBindCol 13-19
    - SQLBindParameter 13-31
    - SQLGetData 13-167
  - supported 13-73, 13-221, 13-222
  - See also* C data types.
  - See also* Converting data.
- SQL identifiers
  - special characters 13-209
- SQL quoted identifiers
  - case-sensitivity 13-208
- SQL statements
  - batch 13-229, B-11
  - direct execution 6-7
  - embedded D-1, D-5
    - ODBC function
      - equivalents D-5, D-6
  - information types 13-190
  - interoperability B-1
  - keywords B-21
  - maximum length 13-202
  - native 13-230
  - qualifier usage in 13-207
  - query timeout 13-294
  - SQLCancel 13-52
  - SQLExecDirect 13-120
  - SQLExecute 13-127
  - SQLPrepare 13-250
- SQLAllocConnect
  - allocating handles 5-5
  - function description 13-11, 13-14, 13-16, 13-18, 13-40, 13-51, 13-54, 13-62, 13-68, 13-77, 13-83, 13-87, 13-92, 13-95, 13-106, 13-110, 13-113, 13-122, 13-129, 13-146, 13-152, 13-154, 13-156, 13-160, 13-163, 13-166, 13-176, 13-183, 13-215, 13-218, 13-227, 13-230, 13-233, 13-235, 13-238, 13-241, 13-246, 13-253, 13-257, 13-265, 13-272, 13-274, 13-283, 13-287, 13-288, 13-297, 13-307, 13-315, 13-321, 15-3, 15-7
  - with SQLConnect 13-80
- SQLAllocEnv
  - function description 13-14, 13-40, 13-51, 13-54, 13-62, 13-68, 13-77, 13-83, 13-87, 13-92, 13-95, 13-106, 13-110, 13-113, 13-122, 13-129, 13-146, 13-152, 13-154, 13-156, 13-160, 13-163, 13-166, 13-176, 13-183, 13-215, 13-218, 13-227, 13-230, 13-233, 13-235, 13-238, 13-241, 13-246, 13-253, 13-257, 13-265, 13-272, 13-274, 13-283, 13-287, 13-287, 13-288, 13-297, 13-307, 13-315, 13-321, 13-327, 15-3, 15-7
  - initializing handles 5-4
  - with SQLConnect 13-80
- SQLAllocStmt 6-5, 13-16
- SQLBindCol
  - binding columns 7-4
  - code example 13-24
  - function description 13-18
  - truncating data 13-22
  - unbinding columns 9-3, 13-23
  - with SQLFetch 13-150
- SQLBindParameter
  - code example 13-39
  - function description 13-25
  - replaces SQLSetParam 13-31
  - sending long data values 6-10
  - unbinding parameters 9-3, 13-156
  - version compatibility 13-287
  - with SQLParamData 13-33, 13-36
  - with SQLParamOptions 13-33, 13-34
  - with SQLPutData 13-36
- SQLBrowseConnect
  - code example 13-48
  - driver manager 13-47
  - function description 13-40, 15-3, 15-7
  - with SQLDisconnect 13-48



- SQLCancel
  - function description 13-51
- SQLColAttributes
  - column attributes 7-4
  - function description 13-54
- SQLColumnPrivileges 13-62
- SQLColumns
  - catalog 6-9
  - code example 13-75
  - function description 13-68
  - intended usage 13-71
- SQLConnect
  - code example 13-81
  - driver manager 13-80
  - function description 13-77
  - with SQLAllocConnect 13-80
  - with SQLAllocEnv 13-80
  - with
    - SQLSetConnectOption 13-80
- SQLDataSources 5-7, 13-83
- SQLDataSourceToDriver 13-53
- SQLDescribeCol 7-4, 13-87
- SQLDisconnect
  - disconnecting 9-4
  - function description 13-92
  - with SQLBrowseConnect 13-48
  - with SQLFreeConnect 13-153
- SQLDriverConnect
  - connecting with 5-6
  - connection strings 5-6
  - dialog boxes 5-7, 5-8
  - driver manager 13-102
  - function description 13-95
  - login interval 13-105
  - odbc.ini 5-7
- SQLDrivers
  - function description 13-106,
    - 13-110, 13-113, 13-122, 13-129,
    - 13-146, 13-152, 13-154, 13-156,
    - 13-160, 13-163, 13-166, 13-176,
    - 13-183, 13-215, 13-218, 13-227,
    - 13-230, 13-233, 13-235, 13-238,
    - 13-241, 13-246, 13-253, 13-257,
    - 13-265, 13-272, 13-274, 13-283,
    - 13-287, 13-288, 13-297, 13-307,
    - 13-315, 13-321, 13-327
  - listing drivers 5-7
- SQLDriverToDataSource 15-7
- SQLError
  - driver manager 13-112
  - function description 13-110
  - See also* Errors.
  - See also* SQLSTATES.
- SQLExecDirect
  - function description 13-113
- SQLExecute
  - function description 13-122
  - with SQLPrepare 13-127
- SQLExtendedFetch
  - code examples 13-142
  - function description 13-129
  - retrieving rowsets 7-9
  - with SQLGetData 7-9
  - with SQLSetPos 13-136
  - with SQLSetStmtOption 13-133
- SQLFetch
  - fetching data 7-5
  - function description 13-146
  - positioning cursor 7-6
  - with SQLBindCol 13-150
  - with SQLGetData 13-150
- SQLFreeConnect
  - freeing connection handles 9-4
  - function description 13-152
  - with SQLDisconnect 13-153
  - with SQLFreeEnv 13-155
- SQLFreeEnv
  - freeing environment handles 9-4
  - function description 13-154
  - with SQLFreeConnect 13-155
- SQLFreeStmt
  - closing cursors 7-6, 13-156
  - freeing statement handles 9-3,
    - 13-159
  - function description 13-156
  - unbinding columns 9-3, 13-23,
    - 13-156
  - unbinding parameters 6-8, 13-30,
    - 13-156
- SQLGetConnectOption 13-160
- SQLGetCursorName 13-163
- SQLGetData
  - code example 13-175
  - function description 13-166
  - interoperability 7-7, 7-9
  - long data values 7-7, 13-22
  - unbound columns 7-7
  - with SQLExtendedFetch 7-9
  - with SQLFetch 13-150
- SQLGetFunctions
  - code example 13-181
  - function description 13-176
- SQLGetInfo
  - code example 13-213
  - data sources 13-189
  - DBMSs 13-188
  - drivers 13-188
  - function description 13-183
  - scalar functions 13-192
  - SQL statements 13-190
- SQLGetStmtOption 13-215
- SQLGetTypeInfo
  - ALTER TABLE statements B-1
  - CREATE TABLE statements B-1
  - function description 13-218
  - supported data types 3-8
- SQLMoreResults 13-227
- SQLNativeSql 13-230
- SQLNumParams 13-233
- SQLNumResultCols 7-4, 13-235
- SQLParamData
  - data-at-execution
    - parameters 6-10
  - function description 13-238
  - with SQLBindParameter 13-33,
    - 13-36
  - with SQLPutData 13-238
- SQLParamOptions
  - code example 13-245
  - function description 13-241
  - multiple parameter values 13-241
  - with SQLBindParameter 13-33,
    - 13-34
- SQLPrepare
  - function description 13-246
  - with SQLExecute 13-127
- SQLPrimaryKeys 6-9, 13-253
- SQLProcedureColumns
  - listing columns 6-18
- SQLProcedures
  - code example 13-263
  - listing procedures 6-18
- SQLPutData
  - code example 13-269
  - data-at-execution
    - parameters 6-10

- function description 13-265
- with SQLBindParameter 13-36
- with SQLParamData 13-238
- SQLRemoveDSNFromIni 14-6
- SQLRowCount
  - function description 13-272
  - interoperability 7-4
- SQLSetConnectOption
  - commit mode 6-9
  - function description 13-274
  - translating data 5-9
  - with SQLConnect 13-80
  - with SQLTransact 13-278
  - See also* Connection options.
- SQLSetCursorName 13-283
- SQLSetParam 13-287
- SQLSetPos
  - locking rows 7-13
  - with SQLExtendedFetch 13-136
- SQLSetScrollOptions 13-287
- SQLSetStmtOption
  - function description 13-288
  - with SQLExtendedFetch 13-133
  - See also* Statement options.
- SQLSpecialColumns 6-10, 13-297
- SQLSTATE
  - guidelines 8-3
  - naming conventions 13-9
  - values A-1
  - values. *See also* specific function description.
- SQLStatistics 6-10, 13-307
- SQLTablePrivileges 13-315
- SQLTables
  - argument syntax 13-324
  - catalog 6-9
  - formatting 13-324
  - function description 13-321
- SQLTransact
  - commit mode 13-278
  - committing 6-9
  - function description 13-327
  - rolling back 6-9
  - two-phase commit 13-329
  - with
    - SQLSetConnectOption 13-278
- SQLWriteDSNToIni 14-6
- SQLWritePrivateProfileString 14-6
- sql.log file 1-11

- SQL\_ERROR 8-3
- SQL\_INVALID\_HANDLE 8-3
- SQL\_NEED\_DATA 8-3
- SQL\_NO\_DATA\_FOUND 8-3
- SQL\_STILL\_EXECUTING 8-3
- SQL\_SUCCESS 8-3
- SQL\_SUCCESS\_WITH\_INFO 8-3
- Statement handles
  - allocating 6-5
  - defined 3-7
  - driver 13-197
  - freeing 9-3, 13-290
  - overwriting 13-18
  - SQLAllocStmt 13-17
  - SQLFreeStmt 13-159
  - SQLMoreResults 13-229
  - with threads 13-18
  - See also* Handles.
- Statement options
  - binding type 13-291
  - concurrency 7-12, 7-13
  - cursor type 13-293
  - driver-specific 13-291
  - maximum data length 13-293
  - maximum length 13-217, 13-291
  - maximum rows 13-294
  - releasing 13-290
  - reserved 13-291
  - retrieving 13-217
  - retrieving data 13-295
  - setting 13-290
  - substituting values 13-291
- Static cursors
  - described 7-10
  - sensitivity 13-210
- Static SQL 10-4
- Statistics, listing 13-310
- Status array
  - errors 13-138
  - SQLExtendedFetch 13-141
- Status information
  - retrieving 8-4
  - See also* Errors.
- String data. *See* Character data.
- String functions
  - supported by INFORMIX
    - driver 1-15
- Strings, connection. *See* Connection strings.

- SYNONYM table type 13-325
- Syntax
  - connection strings 13-45, 13-46, 13-101
  - ODBC SQL 6-12
  - SQL B-11
- System functions
  - supported by INFORMIX
    - driver 1-15
- System requirements
  - for INFORMIX-CLI (UNIX) 2-4
  - for INFORMIX-CLI (Windows) 2-15
- SYSTEM TABLE table type 13-325

---

## T

- Table definition statements
  - qualifier usage in 13-207
- TABLE table type 13-325
- Tables
  - accessibility 13-192
  - cardinality 13-313
  - columns *See* Columns 13-62
  - defined 13-9
  - in SELECT statement 13-203
  - indexes. *See* Indexes.
  - maximum columns in 13-202
  - names
    - maximum length 13-203
    - retrieving 13-311, 13-318
    - special characters 13-209
  - SQLColumnPrivileges 13-66
  - SQLColumns 13-72
  - SQLTables 13-324, 13-325
  - owner name *See* Owner
    - names 13-66
  - pages 13-313
  - primary keys 13-256
  - qualifier *See* Qualifiers 13-66
  - rows. *See* Rows.
  - statistics 13-310
  - term, vendor-specific 13-210
  - types 13-325
  - versus views 13-9
- Termination byte, null. *See* Null-termination byte.

- Terms, vendor-specific
    - owner 13-206
    - procedure 13-206
    - qualifier 13-207
    - table 13-210
  - Threads, multiple
    - with connection handles 13-13
    - with environment handles 13-15
    - with statement handles 13-18
  - Time data
    - converting to SQL C-33
    - literals 6-13
    - specifying conversions
      - SQLBindCol 13-19
      - SQLBindParameter 13-31
      - SQLGetData 13-167
  - Time-out
    - login 13-105
    - query 13-294
  - Timestamp data
    - converting to C C-22
    - converting to SQL C-34
    - literals 6-13
    - specifying conversions
      - SQLBindCol 13-19
      - SQLBindParameter 13-31
      - SQLGetData 13-167
  - TIMESTAMPADD
    - intervals 13-211
  - TIMESTAMPDIFF
    - intervals 13-211
  - Trace file 1-10, 13-279
  - Trace keyword
    - SQLAllocEnv 13-15
  - TraceAutoStop, definition of 1-11
  - TraceFile, definition of 1-10
  - Trace, definition of 1-10
  - Tracing
    - SQLAllocEnv 13-15
    - SQLFreeEnv 13-155
    - SQLSetConnectOption 13-279
  - Traffic, network
    - maximum data length 13-293
    - packet size 13-279
    - prepared statements 6-6
  - Transactions
    - beginning 6-9, 13-330
    - commit mode 6-9, 13-278
    - committing 6-9, 13-329
    - concurrency. *See* Concurrency.
    - cursor behavior 13-330
    - incomplete 13-94
    - interoperability 6-9
    - isolation levels 13-196
    - multiple active 13-203
    - ODBC function equivalents D-8
    - rolling back 6-9, 13-329
    - serializability 7-12
    - SQLExecDirect 13-120
    - SQLExecute 13-127
    - SQLTransact 13-329
    - support of 13-212
    - two-phase commit 13-329
  - Transferring binary data C-8
  - Transitions, state. *See* State transitions.
  - Translation options
    - default 14-7
    - described 13-282
    - specifying 13-280
    - SQLDriverConnect 13-105
  - Translation setup shared library
    - ConfigTranslator 14-3
    - See also* Translation shared libraries.
  - Translation shared libraries
    - character sets 5-9
    - default 13-105
    - default option 14-7
    - described 5-9, 13-282
    - function summary 12-10
    - setup shared library. *See* Translation setup shared library.
    - specifying 13-280
    - SQLDataSourceToDriver 15-6
    - SQLDriverConnect 13-105
    - SQLDriverToDataSource 15-10
    - truncating data 15-6, 15-10
    - See also* Converting data.
    - See also* odbcnst.ini.
  - Truncating data
    - maximum data length 13-294
    - output buffers 3-7
    - SQLBindCol 13-22
    - SQLBindParameter 13-34
    - SQLFetch 13-150
    - SQLGetData 13-174
  - translation shared libraries 15-6, 15-10
    - See also* Binary data.
    - See also* Character data.
  - Two-phase commit 13-329
  - Typographical
    - Conventions Intro-8
- 
- ## U
- Unbinding columns 13-23, 13-156
  - Unbinding parameters 13-156
  - Underscore (\_) 13-10
  - UNION clauses 13-213
  - Unloading drivers 13-81
  - UPDATE statements
    - affected rows 13-272
    - bulk 13-241
    - privileges 13-67
    - qualifier usage in 13-207
    - See also* SQLSetPos.
  - Updates, positioned. *See* Positioned UPDATE statements.
  - User names
    - maximum length 13-203
    - retrieving 13-213
- 
- ## V
- Values
    - procedure return 13-31
  - Version
    - DBMS 13-195
    - driver
      - number 13-197
      - ODBC version 13-197
    - driver manager 13-205
  - Version compatibility
    - default C data types C-6
    - information types 13-187
    - number of input
      - parameters 13-261
    - number of output
      - parameters 13-261
    - number of result sets 13-261
    - parameter binding 13-287
    - positioned statements 7-14

- SQLBindParameter 13-179, 13-287
- SQLForeignKeys 13-151
- SQLSetParam 13-179, 13-287
- Versioning isolation level 13-196
- VIEW table type 13-325
- Views 13-9

---

## W

- Window handles
  - null 14-5
  - parent, ConfigDSN 14-5
  - quiet mode 13-280
  - See also* SQLDriverConnect.
- Windows 95
  - ODBC.INI 1-7, 2-17
- Windows NT
  - ODBC.INI 1-7, 2-17
- Windows NT registry
  - data sources
    - adding 14-5
    - configuring 14-6
    - removing 14-7

---

## Symbols

- % (Percent sign) 13-10, 13-324
- \* (Asterisk) 13-46
- ? (Question mark)
  - browse result connection
    - string 13-46
  - parameter markers 13-120, 13-250
- \_ (Underscore) 13-10
- { } (braces) 13-46