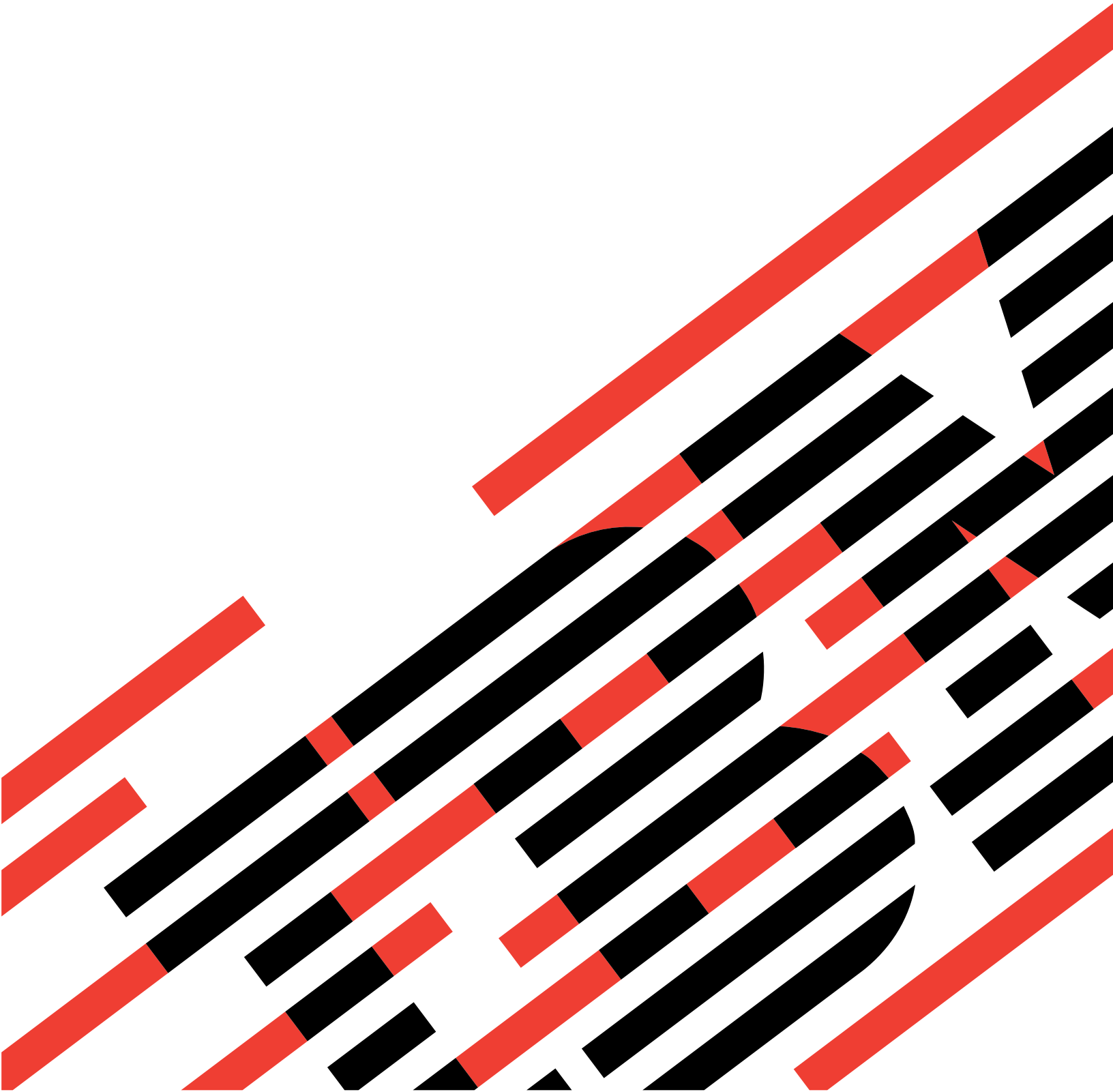
IBM

eserver

iSeries

DB2 通用数据库 iSeries 版 SQL 调用层接口（ODBC）

版本 5



IBM

@server

iSeries

DB2 通用数据库 iSeries 版 SQL 调用层接口（ODBC）

版本 5

目录

关于 DB2 通用数据库 iSeries 版 SQL 调用层接口（ODBC）

本信息提供对典型 DB2 UDB CLI 应用程序的概述。本书包含下列信息:

- 介绍了 DB2 UDB CLI 并讨论此接口的背景以及它与嵌入式 SQL 的关系。
- 讨论了 DB2 UDB CLI 应用程序中的任务或步骤，并介绍了概念、函数以及它们之间的交互作用。
- 构成 DB2 UDB CLI 的函数的参考信息。
- 包含下列附录:
  - 第 241 页的附录 A, 『DB2 UDB CLI 一般诊断信息』，此附录包含在整本书中引用的表。
  - 第 243 页的附录 B, 『DB2 UDB CLI 包含文件』，此附录列示所有 DB2 UDB CLI 应用程序都包括的头文件。
  - 第 261 页的附录 C, 『示例 DB2 UDB CLI 应用程序代码列表』，此附录列示在整本书中使用的示例代码段的完整源代码。
  - 第 271 页的附录 D, 『以服务器方式运行 DB2 UDB CLI』，此附录包含有关如何使用 CLI 应用程序来为多个用户提供服务的信息。

有关本指南的更多信息，参见下列主题:

- 『谁应该阅读 DB2 UDB iSeries 版 SQL 调用层接口（ODBC）一书』
- 『DB2 UDB iSeries 版 SQL 调用层接口（ODBC）中对 V5R2 的新增内容』
- 第 viii 页的『代码不保证声明信息』

然后，请阅读第 1 页的第 1 章，『CLI 简介』以便入门。

谁应该阅读 DB2 UDB iSeries 版 SQL 调用层接口（ODBC）一书

本书是为具备 SQL 以及 C 编程语言的知识并且想使用 DB2 UDB CLI 函数来调用动态 SQL 语句的应用程序员编写的。

DB2 UDB iSeries 版 SQL 调用层接口（ODBC）中对 V5R2 的新增内容

本发行版添加了下列 API:

- SQLColumnPrivileges — 获取与表的列相关联的特权
- SQLNextResult — 处理下一个结果集
- SQLTablePrivileges — 获取与表相关联的特权

本发行版更新了下列 API:

- SQLBindParam — 将缓冲区绑定到参数标记
- SQLColAttributes — 列属性
- SQLEndTran — 落实或回滚事务
- SQLGetConnectOption — 返回连接选项的当前设置
- SQLGetInfo — 获取一般信息
- SQLGetLength — 检索字符串值的长度
- SQLGetStmtOption — 返回语句选项的当前设置
- SQLProcedureColumns — 获取过程的输入／输出参数信息

- SQLProcedures — 获取过程名列表
- SQLSetConnectAttr — 设置连接属性
- SQLSetDescRec — 设置描述符记录
- SQLSetEnvAttr — 设置环境属性
- SQLSetStmtAttr — 设置语句属性
- SQLTables — 获取表信息

已更新『CLI 简介』和『编写 DB2® CLI 应用程序』中的信息。

代码不保证声明信息

本文档包含编程示例。

IBM® 授予您非专用版权许可证来使用所有编程代码示例，您可以根据它们生成针对您的特定需要进行裁制的类似功能。

所有样本代码都是由 IBM 提供，仅供说明。尚未在所有环境下全面测试这些示例。因此，IBM 不能保证或暗示这些程序的可靠性、可服务性或功能。

此处包含的所有程序均是以“按现状”的基础提供的，不附有任何形式的保证。明确拒绝有关非侵权性、适销性和适用于某特定用途的默示保证。

第 1 章 CLI 简介

“DB2 UDB 调用层接口”（CLI）是一个可调用的“结构化查询语言”（SQL）编程接口，在除 DB2 UDB zOS 版和 OS/390® 版以及 DB2 服务器 VSE 版和 VM 版之外的所有 DB2 环境中都支持此接口。可调用的 SQL 接口是一种 WinSock 应用程序接口（API），它用于使用函数调用来启动动态 SQL 语句的数据库访问。

DB2 UDB CLI 可用来替代嵌入式动态 SQL。嵌入式动态 SQL 与 DB2 UDB CLI 之间的重要差别是 SQL 语句的启动方式不相同。在 iSeries 上，此接口可用于任何 ILE 语言。

DB2 UDB CLI 还提供了全面的“第 1 级 Microsoft® 开放式数据库连接（ODBC）”支持以及许多“第 2 级”功能。对于大多数部件而言，ODBC 是 ANS 和 ISO SQL CLI 标准的超集。

有关更多信息，参见:

- 『DB2 UDB CLI 背景信息』
- 『调用层接口』
- 第 3 页的『DB2 UDB CLI 与嵌入式 SQL 之间的差别』

DB2 UDB CLI 背景信息

理解 DB2 UDB CLI 或任何可调用的 SQL 接口的基础并将其与现有接口作比较十分重要。

ISO 标准 9075:1999 — “数据库语言 SQL 第 3 部分: 调用层接口”提供了 CLI 的标准定义。此接口的目的是通过使应用程序能够独立于任何一个数据库服务器来提高应用程序的可移植性。

ODBC 提供了 Driver Manager for Windows®，这个程序为每个 ODBC 驱动程序提供了中央控制点（一个实现了 ODBC 函数调用并且与特定 DBMS 进行交互的动态链接库（DLL））。

调用层接口

下列调用层接口 API 可用于 iSeries 上的数据库访问:

- **连接**
  - 第 68 页的『SQLConnect — 连接到数据源』
  - 第 71 页的『SQLDataSources — 获取数据源列表』
  - 第 80 页的『SQLDisconnect — 与数据源断开连接』
  - 第 82 页的『SQLDriverConnect —（扩充）连接到数据源』
- **诊断**
  - 第 88 页的『SQLError — 检索错误信息』
  - 第 133 页的『SQLGetDiagField — 返回诊断信息（可扩展）』
  - 第 136 页的『SQLGetDiagRec — 返回诊断信息（简洁）』
- **元数据**
  - 第 65 页的『SQLColumns — 获取表的列信息』
  - 第 62 页的『SQLColumnPrivileges — 获取与表列相关联的特权』
  - 第 105 页的『SQLForeignKeys — 获取外键列的列表』
  - 第 143 页的『SQLGetInfo — 获取一般信息』

- 第 122 页的『SQLGetConnectOption — 返回连接选项的当前设置』
- 第 123 页的『SQLGetCursorName — 获取游标名』
- 第 127 页的『SQLGetData — 从列中获取数据』
- 第 128 页的『SQLGetDescField — 获取描述符字段』
- 第 131 页的『SQLGetDescRec — 获取描述符记录』
- 第 139 页的『SQLGetEnvAttr — 返回环境属性的当前设置』
- 第 140 页的『SQLGetFunctions — 获取函数』
- 第 143 页的『SQLGetInfo — 获取一般信息』
- 第 155 页的『SQLGetLength — 检索字符串值的长度』
- 第 157 页的『SQLGetPosition — 返回字符串的开始位置』
- 第 160 页的『SQLGetStmtAttr — 获取语句属性的值』
- 第 162 页的『SQLGetStmtOption — 返回语句选项的当前设置』
- 第 163 页的『SQLGetSubString — 检索字符串值的一部分』
- 第 166 页的『SQLGetTypeInfo — 获取数据类型信息』
- 第 206 页的『SQLSetConnectAttr — 设置连接属性』
- 第 210 页的『SQLSetConnectOption — 设置连接选项』
- 第 212 页的『SQLSetCursorName — 设置游标名』
- 第 214 页的『SQLSetDescField — 设置描述符字段』
- 第 216 页的『SQLSetDescRec — 设置描述符记录』
- 第 218 页的『SQLSetEnvAttr — 设置环境属性』
- 第 222 页的『SQLSetParam — 设置参数』
- 第 223 页的『SQLSetStmtAttr — 设置语句属性』
- 第 226 页的『SQLSetStmtOption — 设置语句选项』

• **使用句柄**

- 第 24 页的『SQLAllocConnect — 分配连接句柄』
- 第 27 页的『SQLAllocEnv — 分配环境句柄』
- 第 30 页的『SQLAllocHandle — 分配句柄』
- 第 32 页的『SQLAllocStmt — 分配语句句柄』
- 第 70 页的『SQLCopyDesc — 复制描述语句』
- 第 110 页的『SQLFreeConnect — 释放连接句柄』
- 第 111 页的『SQLFreeEnv — 释放环境句柄』
- 第 112 页的『SQLFreeHandle — 释放句柄』
- 第 114 页的『SQLFreeStmt — 释放（或重设）语句句柄』
- 第 203 页的『SQLReleaseEnv — 释放所有环境资源』

DB2 UDB CLI 与嵌入式 SQL 之间的差别

使用嵌入式 SQL 接口的应用程序需要预编译器来将 SQL 语句转换成代码。将对代码进行编译、绑定到数据库并执行。相反，DB2 UDB CLI 应用程序不要求进行预编译或绑定，而是使用标准的函数集来在运行时执行 SQL 语句和相关服务。

由于预编译器传统上是特定于数据库产品的（这实际上是将应用程序束缚在该产品上），所以这种差别十分重要。DB2 UDB CLI 使您能够编写独立于任何特定数据库产品的可移植应用程序。这种独立性意味着不必重新编译或重新绑定 DB2 UDB CLI 应用程序就能够访问不同的数据库产品。应用程序将在运行时选择适当的数据库产品。

DB2 UDB CLI 与嵌入式 SQL 在下列方面也不相同:

- DB2 UDB CLI 不要求显式地声明游标。DB2 UDB CLI 根据需要生成它们。然后，应用程序可以在正常游标取装模型中将所生成的游标用于多个行 SELECT 语句以及定位型 UPDATE 和 DELETE 语句。
- 在 DB2 UDB CLI 中，OPEN 语句不是必需的。SELECT 的执行就可以自动导致游标打开。
- 与嵌入式 SQL 不同，DB2 UDB CLI 允许在 EXECUTE IMMEDIATE 语句的等价项（SQLExecDirect() 函数）上使用参数标记。
- 在 DB2 UDB CLI 中，COMMIT 或 ROLLBACK 是通过 SQLTransact() 或 SQLEndTran() 函数调用发出的，而不是通过将其作为 SQL 语句传送来发出的。
- DB2 UDB CLI 代替应用程序管理与语句相关的信息，并提供了***语句句柄***来将该信息作为抽象对象进行引用。此句柄使得应用程序不需要使用特定于产品的数据结构。
- 与语句句柄类似，***环境句柄***和***连接句柄***提供了一种引用所有全局变量以及特定于连接的信息的方法。
- DB2 UDB CLI 使用 X/Open SQL CAE 规范定义的 SQLSTATE 值。虽然格式以及许多值与 IBM 关系数据库产品使用的值一致，但还是存在差别。

尽管存在这些差别，但是嵌入式 SQL 与 DB2 UDB CLI 之间还是存在着重要的公共概念:

DB2 UDB CLI 可以执行任何可以在嵌入式 SQL 中动态准备的 SQL 语句。由于 DB2 UDB CLI 并不是实际地*执行* SQL 语句本身，而是将其传送至 DBMS 以便进行动态执行，所以能够保证这一点。

表 1 列示了每个 SQL 语句以及是否可使用 DB2 UDB CLI 来执行它。

表 1. SQL 语句

SQL 语句	Dyn [a]	CLI [c]
ALTER TABLE	X	X
BEGIN DECLARE SECTION [b]		
CALL	X	X
CLOSE		SQLFreeStmt()
COMMENT ON	X	X
COMMIT	X	SQLTransact(), SQLEndTran()
CONNECT（第 1 类）		SQLConnect()
CONNECT（第 2 类）		SQLConnect()
CREATE INDEX	X	X
CREATE TABLE	X	X
CREATE VIEW	X	X
DECLARE CURSOR [b]		SQLAllocStmt()
DELETE	X	X
DESCRIBE		SQLDescribeCol(), SQLColAttributes()
DISCONNECT		SQLDisconnect()
DROP	X	X
END DECLARE SECTION [b]		
EXECUTE		SQLExecute()

表 1. SQL 语句 (续)

SQL 语句	Dyn [a]	CLI [c]
EXECUTE IMMEDIATE		SQLExecDirect()
FETCH		SQLFetch()
GRANT	X	X
INCLUDE [b]		
INSERT	X	X
LOCK TABLE	X	X
OPEN		SQLExecute() 和 SQLExecDirect()
PREPARE		SQLPrepare()
RELEASE		SQLDisconnect()
REVOKE	X	X
ROLLBACK	X	SQLTransact() 和 SQLEndTran()
SELECT	X	X
SET CONNECTION		
UPDATE	X	X
WHENEVER [b]		

注:

[a] **Dyn** 表示动态。此列表中的所有语句都可以编码为静态 SQL，但只有那些标有 **X** 的语句才可以编码为动态 SQL。

[b] 这是不可执行的语句。

[c] X 指示可使用 SQLExecDirect() 或者 SQLPrepare() 和 SQLExecute() 来执行此语句。如果有等价的 DB2 UDB CLI 函数，则列示了函数名。

每个 DBMS 都可能有附加的可以动态准备的语句，在这种情况下，DB2 UDB CLI 将它们传送至 DBMS。这里有一个例外，一些 DBMS 动态地准备 COMMIT 和 ROLLBACK，但不传送它们。相反，应使用 SQLTransact() 或 SQLEndTran() 来指定 COMMIT 或 ROLLBACK。

有关其它信息，参见:

- 『使用 DB2 UDB CLI 来代替嵌入式 SQL 的优点』
- 第 6 页的『在 DB2 UDB CLI、动态 SQL 与静态 SQL 之间作决定』

使用 DB2 UDB CLI 来代替嵌入式 SQL 的优点

相对于嵌入式 SQL，DB2 UDB CLI 接口具有若干项关键的优点。

- 它相当适合于客户机／服务器环境，在此环境中，在构建应用程序时目标数据库是未知的。它提供了用于执行 SQL 语句的一致接口，而无须考虑应用程序连接到哪个数据库服务器。
- 它通过除去对预编译器的依赖性来提高应用程序的可移植性。应用程序不是作为经过编译的应用程序或运行时库分发的，而是作为针对每一数据库产品进行了预处理的源代码分发的。
- DB2 UDB CLI 应用程序不必绑定到与它们所连接的每一个数据库。
- DB2 UDB CLI 应用程序可以同时连接到多个数据库。
- 与嵌入式 SQL 应用程序不同，DB2 UDB CLI 应用程序并不负责控制全局数据区，如 SQLCA 和 SQLDA。相反，DB2 UDB CLI 分配并控制必需的数据结构，并为应用程序提供了*句柄*来引用它们。

在 DB2 UDB CLI、动态 SQL 与静态 SQL 之间作决定

您应该选择哪个接口取决于您的应用程序。

DB2 UDB CLI 相当适合于那些需要可移植性而不需要由特定 DBMS 提供的 API 或实用程序（例如，目录数据库、备份和恢复）的基于查询的应用程序。这并不表示使用 DB2 UDB CLI 将从应用程序中调用特定于 DBMS 的 API。这意味着应用程序将不再可移植。

另一个重要的注意事项是动态 SQL 与静态 SQL 之间的性能比较。动态 SQL 是在运行时准备的，而静态 SQL 是在预编译阶段准备的。由于准备语句需要附加的处理时间，所以静态 SQL 可能更有效率。如果您选择静态 SQL 而不选择动态 SQL，则 DB2 UDB CLI 就不是一个选项。

在大多数情况下，在接口之间所作的选择是对个人喜好开放的。您过去的经验可能会使一个备选项比另一个备选项更为直观。

第 2 章 编写 DB2 UDB CLI 应用程序

DB2 UDB CLI 应用程序由一组任务组成，而每个任务都由一组离散步骤组成。当应用程序运行时，应用程序可能会执行其它任务。应用程序调用一个或多个 DB2 UDB CLI 函数来执行这些任务中的每一个。

每个 DB2 UDB CLI 应用程序都包含图 1 所示的三个主任务。如果不按照图中所示的顺序来调用函数，就会导致错误。

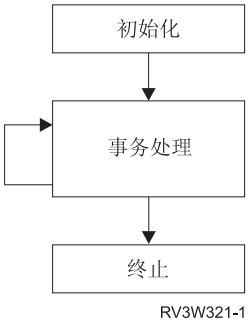


图 1. DB2 UDB CLI 应用程序的概念视图

初始化 此任务分配并初始化一些资源，以便为*事务处理*主任务作准备。有关详细信息，请参考第 8 页的『DB2 UDB CLI 应用程序中的初始化和终止任务』。

事务处理

这是应用程序的主任务。为了查询和修改 SQL，将把语句传送至 DB2 UDB CLI。有关详细信息，请参考第 10 页的『DB2 UDB CLI 应用程序中的事务处理任务』。

终止 此任务释放已分配的资源。资源通常由通过唯一句柄标识的数据区组成。在释放资源之后，其它任务可使用这些句柄。有关详细信息，请参考第 8 页的『DB2 UDB CLI 应用程序中的初始化和终止任务』。

除了上面列示的三个任务之外，应用程序中还会发生一般性的任务，如处理诊断消息。

在本主题中，提供了示例来举例说明如何在 DB2 UDB CLI 应用程序中使用这些函数。

有关其它信息，参见:

- 第 8 页的『DB2 UDB CLI 应用程序中的初始化和终止任务』
- 第 10 页的『DB2 UDB CLI 应用程序中的事务处理任务』
- 第 15 页的『DB2 UDB CLI 应用程序中的诊断』
- 第 16 页的『DB2 UDB CLI 函数中的数据类型和数据转换』
- 第 18 页的『在 DB2 UDB CLI 函数中使用字符串自变量』

有关每个函数的完整描述及用法信息，请参考第 21 页的第 3 章，『DB2 UDB CLI 函数』。

DB2 UDB CLI 应用程序中的初始化和终止任务

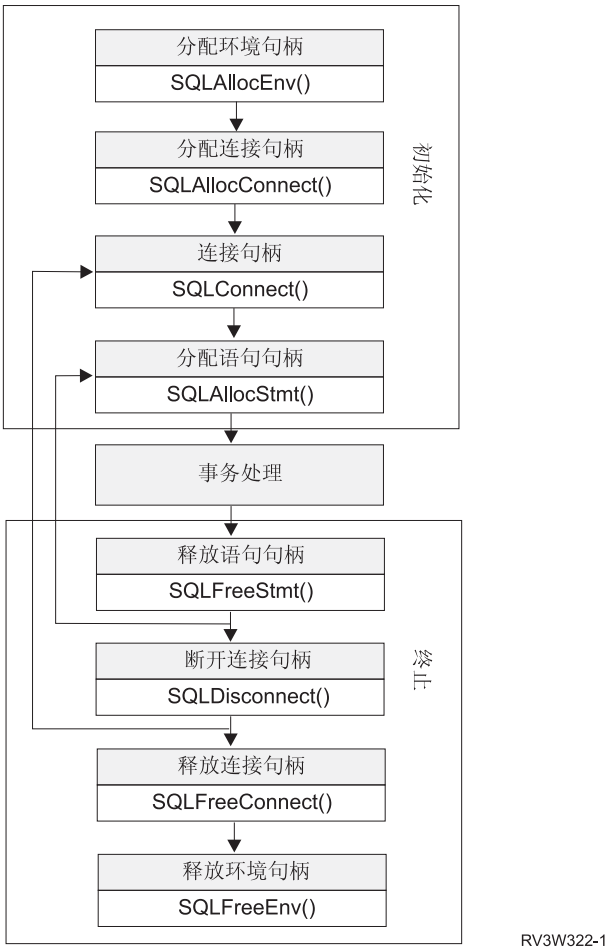


图 2. 初始化和终止任务的概念视图

图 2 显示了初始化和终止任务的函数调用序列。第 11 页的图 3 显示了位于图中央的事务处理任务。

初始化任务分配并初始化环境句柄和连接句柄。终止任务将它们释放。句柄是用来引用由 DB2 UDB CLI 控制的数据对象的变量。通过使用句柄，应用程序就不必分配和管理全局变量或数据结构，如 IBM DBMS 的嵌入式 SQL 接口中使用的 SQLDA 或 SQLCA。于是，应用程序在调用其它 DB2 UDB CLI 函数时，它传送适当的句柄。共有三种类型的句柄:

环境句柄

环境句柄是指包含关于应用程序状态的全局信息的数据对象。此句柄是通过调用 `SQLAllocEnv()` 分配的，并且是通过调用 `SQLFreeEnv()` 释放的。在可以分配连接句柄之前，必须分配环境句柄。每个应用程序只能分配一个环境句柄。

连接句柄

连接句柄是指包含与 DB2 UDB CLI 管理的连接相关联的信息的数据对象。这包括一般状态信息、事务状态和诊断信息。每个连接句柄都是通过调用 `SQLAllocConnect()` 分配的，并且是通过调用 `SQLFreeConnect()` 释放的。应用程序必须为每一个与数据库服务器的连接分配连接句柄。

语句句柄

语句句柄在下一个任务中讨论。

参见『示例: DB2 UDB CLI 应用程序中的初始化和连接』。

示例: DB2 UDB CLI 应用程序中的初始化和连接

有关代码示例的信息，参见第 viii 页的『代码不保证声明信息』。

```
/*****************************************************
** file = basiccon.c
**    - demonstrate basic connection to two datasources.
**    - error handling  ignored for simplicity
**
**  Functions used:
**
**    SQLAllocConnect  SQLDisconnect
**    SQLAllocEnv      SQLFreeConnect
**    SQLConnect       SQLFreeEnv
**
**
*****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "sqlcli.h"

int
connect(SQLHENV henv,
        SQLHDBC * hdbc);

#define MAX_DSN_LENGTH    18
#define MAX_UID_LENGTH    10
#define MAX_PWD_LENGTH    10
#define MAX_CONNECTIONS   5

int
main()
{
    SQLHENV         henv;
    SQLHDBC         hdbc[MAX_CONNECTIONS];

    /* allocate an environment handle   */
    SQLAllocEnv(&henv);

    /* Connect to first data source */
    connect(henv, &hdbc[0];);

    /* Connect to second data source */
    connect(henv, &hdbc[1];);

    /*********   Start Processing Step  ************************/
    /* allocate statement handle, execute statement, and so forth       */
    /*********   End Processing Step  *************************/

    printf("\nDisconnecting .....\n");
    SQLDisconnect(hdbc[0]);     /* disconnect first connection   */
    SQLDisconnect(hdbc[1]);     /* disconnect second connection  */
    SQLFreeConnect(hdbc[0]);    /* free first connection handle  */
    SQLFreeConnect(hdbc[1]);    /* free second connection handle */
    SQLFreeEnv(henv);           /* free environment handle       */

    return (SQL_SUCCESS);
}

/*********************************************************************
**   connect - Prompt for connect options and connect              **
*********************************************************************/

int
```

```
connect(SQLHENV henv,
        SQLHDBC * hdbc)
{
    SQLRETURN       rc;
    SQLCHAR         server[MAX_DSN_LENGTH + 1], uid[MAX_UID_LENGTH + 1],
pwd[MAX_PWD_LENGTH
+ 1];
    SQLCHAR         buffer[255];
    SQLSMALLINT     outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    SQLAllocConnect(henv, hdbc);/* allocate a connection handle     */

    rc = SQLConnect(*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database\n");
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
        return (SQL_SUCCESS);
    }
}
```

DB2 UDB CLI 应用程序中的事务处理任务

下图显示了 DB2 UDB CLI 应用程序中函数调用的典型次序。这并没有显示所有的函数或可能的路径。

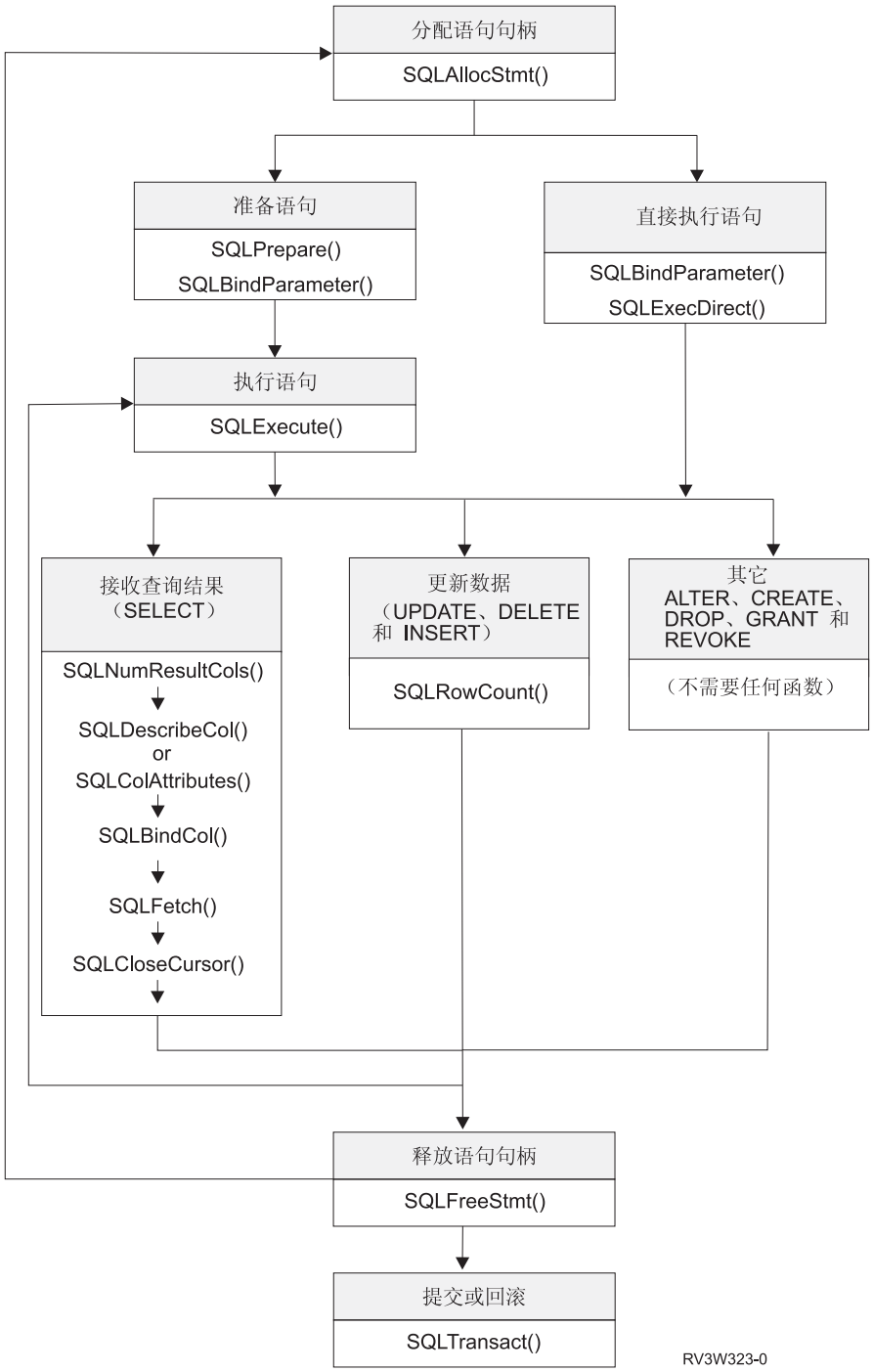


图 3. 事务处理

图 3 显示了事务处理任务中的步骤和 DB2 UDB CLI 函数。此任务包含 5 个步骤:

- 第 12 页的『在 DB2 UDB CLI 应用程序中分配语句句柄』
- 第 12 页的『DB2 UDB CLI 应用程序中的准备和执行任务』
- 第 13 页的『在 DB2 UDB CLI 应用程序中处理结果』
- 第 14 页的『在 DB2 UDB CLI 应用程序中释放语句句柄』
- 第 14 页的『DB2 UDB CLI 应用程序中的落实或回滚』

需要函数 SQLAllocStmt 来获取将用来处理 SQL 语句的语句句柄。可使用两种语句执行方法。通过使用 SQLPrepare 和 SQLExecute，程序可以将过程分成两个步骤。函数 SQLBindParameter 用来将程序地址绑定到在已准备的 SQL 语句中使用的主变量。第二种方法是直接执行方法，在此方法中，使用对 SQLExecDirect 的单个调用来替换 SQLPrepare 和 SQLExecute。

在执行语句之后，其余的处理取决于 SQL 语句的类型。对于 SELECT 语句，程序使用 SQLNumResultCols、SQLDescribeCol、SQLBindCol、SQLFetch 和 SQLCloseCursor 之类的函数来处理结果集。对于更新数据的语句，可使用 SQLRowCount 来确定受影响的行数。对于其它类型的 SQL 语句，执行语句之后便完成处理。在所有情况下，接着都使用 SQLFreeStmt 来指示不再需要该句柄。

在 DB2 UDB CLI 应用程序中分配语句句柄

SQLAllocStmt() 分配语句句柄。语句句柄是指包含关于 DB2 UDB CLI 管理的 SQL 语句的信息的数据对象。这包括诸如动态自变量、游标信息、动态自变量和列的绑定、结果值以及状态信息之类的信息（这些信息在后面讨论）。每个语句句柄都与连接句柄相关联。

分配语句句柄以便运行语句。能够同时分配的句柄的总数限制是 80000。此限制适用于所有类型的句柄，包括由实现代码隐式分配的描述符句柄。还存在一个远程连接的语句句柄数不得超过 500 的限制。

DB2 UDB CLI 应用程序中的准备和执行任务

在分配语句句柄之后，有两种方法来指定和执行 SQL 语句:

1. 先准备，然后执行:
   a. 使用 SQL 语句作为自变量来调用 SQLPrepare()。
   b. 如果 SQL 语句包含*参数标记*，则调用 SQLSetParam()。
   c. 调用 SQLExecute()
2. 直接执行:
   a. 如果 SQL 语句包含*参数标记*，则调用 SQLSetParam()。
   b. 使用 SQL 语句作为自变量来调用 SQLExecDirect()。

第一种方法将语句的准备与执行分割开来。在下列情况下使用此方法:

- 语句需要重复地执行（通常使用不同的参数值来执行）。这将避免必须多次准备同一个语句。
- 在执行语句之前，应用程序需要关于结果集中的列的信息。

第二种方法将准备步骤与执行步骤合二为一。在下列情况下使用此方法:

- 语句执行一次。这将避免必须调用两个函数来执行该语句。
- 在执行语句之前，应用程序不需要关于结果集中的列的信息。

在 DB2 UDB CLI 应用程序中绑定 SQL 语句中的参数

两种执行方法都允许使用参数标记来代替 SQL 语句中的*表达式*（或嵌入式 SQL 中的主变量）。

参数标记由 “?” 字符表示，它指示 SQL 语句中的一个位置，执行语句时，将在该位置替代应用程序变量的内容。标记是从 1 开始从左到右按顺序引用的。

当应用程序变量与参数标记相关联时，它便*绑定*到参数标记。使用下列各项，通过调用 SQLSetParam() 函数来执行绑定:

- 参数标记的编号
- 指向应用程序变量的指针

- 参数的 SQL 类型
- 变量的数据类型和长度

由于调用 SQLSetParam() 时只传送指针，所以将应用程序变量称为*延迟*自变量。在执行语句之前，不从变量中读取数据。这适用于缓冲区自变量以及指示缓冲区中的数据的长度的自变量。延迟自变量使应用程序能够修改所绑定的参数变量的内容以及使用新的值来重复执行语句。

在调用 SQLSetParam() 时，有可能绑定具有不同于 SQL 语句所需的类型的变量。在这种情况下，DB2 UDB CLI 将把所绑定的变量的内容转换为正确的类型。例如，SQL 语句可能需要整数值，但应用程序把整数当成字符串使用。可以将字符串绑定到该参数，执行语句时，DB2 UDB CLI 将把该字符串转换为整数。有关数据转换的更多信息，请参考第 16 页的『DB2 UDB CLI 函数中的数据类型和数据转换』。

有关更多信息和示例，请参考:

- 第 186 页的『SQLPrepare — 准备语句』
- 第 222 页的『SQLSetParam — 设置参数』
- 第 93 页的『SQLExecute — 执行语句』
- 第 91 页的『SQLExecDirect — 直接执行语句』

在 DB2 UDB CLI 应用程序中处理结果

在执行语句之后，下一个步骤取决于 SQL 语句的类型。

在 DB2 UDB CLI 应用程序中处理 SELECT 语句

如果语句是 SELECT，则通常需要执行下列步骤来检索结果集的每一行:

1. 建立结果集的结构、列数、列类型和长度
2. （可选）将应用程序变量绑定到列以便接收数据
3. 重复地取装下一行数据，并将其接收到绑定的应用程序变量中
4. （可选）在成功地进行每次取装之后，可通过调用 SQLGetData() 来检索先前未绑定的列

注: 上面每个步骤都要求进行一些诊断检查。

第一个步骤要求分析所执行的或准备的语句。如果 SQL 语句是由应用程序生成的，则此步骤不是必需的。这是由于应用程序了解结果集的结构以及每个列的数据类型。如果在运行时生成 SQL 语句（例如，由用户输入），则应用程序需要查询:

- 列数
- 每个列的类型
- 结果集中的每个列的名称

在准备语句之后，或在执行语句之后，可通过调用 SQLNumResultCols() 和 SQLDescribeCol()（或者 SQLColAttributes()）来获取此信息。

第二个步骤使应用程序在下次调用 SQLFetch() 时能够将列数据直接检索到应用程序变量中。对于要检索的每一列，应用程序调用 SQLBindCol() 来将一个应用程序变量绑定到结果集中的一个列。与使用 SQLSetParam() 来绑定到参数标记的变量相似，列也是使用延迟自变量来绑定的。这次，这些变量是输出自变量，并且在调用 SQLFetch() 时将数据写至这些变量。SQLGetData() 也可用来检索数据，因此调用 SQLBindCol() 是可选的。

第三个步骤是调用 SQLFetch() 来取装结果集的第一行或下一行。如果已绑定任何列，则将更新应用程序变量。如果 SQLBindCol 调用上指定的数据类型指示了任何数据转换，则调用 SQLFetch() 时将进行转换。有关数据转换的说明，请参考第 16 页的『DB2 UDB CLI 函数中的数据类型和数据转换』。

最后一个（可选）步骤是调用 SQLGetData() 来检索任何先前未绑定的列。可以使用这种方法来检索所有的列（倘若未将它们绑定的话），也可以将两种方法结合使用。SQLGetData() 对于检索小块的变长列而言也非常有用，而这是无法通过绑定列完成的。与 SQLBindCol() 相同，还可以在这里指示数据转换。有关更多信息，请参考第 16 页的『DB2 UDB CLI 函数中的数据类型和数据转换』。

有关更多信息和示例，请参考:

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 58 页的『SQLColAttributes — 列属性』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 127 页的『SQLGetData — 从列中获取数据』
- 第 180 页的『SQLNumResultCols — 获取结果列数』

在 DB2 UDB CLI 应用程序中处理 UPDATE、DELETE 和 INSERT 语句

如果语句的作用是修改数据（UPDATE、DELETE 或 INSERT），则除了正常的诊断消息检查之外不需要执行任何其它操作。在此情况下，可使用 SQLRowCount() 来获取受 SQL 语句影响的行数。有关更多信息，请参考第 180 页的『SQLNumResultCols — 获取结果列数』。

如果 SQL 语句是定位型 UPDATE 或 DELETE，则有必要使用*游标*。游标是指向 SELECT 语句的结果表中的某一行的可移动指针。在嵌入式 SQL 中，游标用来检索、更新或删除行。在使用 DB2 UDB CLI 时，由于会自动生成一个游标，所以没有必要定义游标。

对于定位型 UPDATE 或 DELETE 语句，需要在 SQL 语句中指定游标的名称。您可以使用 SQLSetCursorName() 来定义自己的游标名，也可以使用 SQLGetCursorName() 来查询所生成的游标的名称。最好使用所生成的名称，这是因为所有错误消息都将引用此名称，而不是引用 SQLSetCursorName() 定义的那个名称。

在 DB2 UDB CLI 应用程序中处理其它 SQL 语句

如果语句既不查询也不修改数据，则除了正常的诊断消息检查之外，不执行任何进一步的操作。

在 DB2 UDB CLI 应用程序中释放语句句柄

调用 SQLFreeStmt() 来对特定语句句柄结束处理。此函数可用来执行下列其中一项或多项操作:

- 取消绑定所有列
- 取消绑定所有参数
- 关闭任何游标并废弃结果
- 删除语句句柄，并释放所有相关联的资源

如果不将语句句柄删除，则可以重新使用它。

DB2 UDB CLI 应用程序中的落实或回滚

最后一个步骤是使用 SQLTransact() 来落实或回滚*事务*。

事务是可恢复的工作单元，或者是一组可以将它们视为一个原子的操作的 SQL 语句。这表示将完成（落实）或撤销（回滚）组中的所有操作，就象它们是单个操作一样。

当使用 DB2 UDB CLI 时，使用 SQLPrepare()、SQLExecDirect() 或 SQLGetTypeInfo() 第一次访问数据库时将隐式地启动事务。当使用 SQLTransact() 来回滚或落实事务时，事务结束。这表示将把其间执行的任何 SQL 语句视为一个工作单元。

何时在 DB2 UDB CLI 应用程序中调用 SQLTransact()

在决定何时结束事务时，请考虑下列各项:

- 只能落实或回滚当前事务，因此请将相关的语句放在同一个事务中。
- 当有未完成的事务时，会挂起各种锁。结束事务将释放锁，并允许其他用户访问数据。这对于所有 SQL 语句（包括 SELECT 语句）都成立。
- 在成功地落实或回滚事务之后，可以从系统作业记录完全地恢复该事务（这取决于 DBMS）。开放事务不可恢复。

在 DB2 UDB CLI 应用程序中调用 SQLTransact() 的效果

当事务结束后:

- 必须准备所有语句，然后才能再次使用它们。
- 在各事务之间维护游标名、绑定的参数和列绑定。
- 关闭所有打开的游标。

有关更多信息和示例，请参考第 239 页的『SQLTransact — 事务管理』。

DB2 UDB CLI 应用程序中的诊断

诊断是指处理应用程序中生成的警告或错误情况。在调用 DB2 UDB CLI 函数时，共有两个级别的诊断:

- 『DB2 UDB CLI 应用程序的返回码』
- 『DB2 UDB CLI SQLSTATE』（诊断消息）

有关错误处理的示例，请参考第 88 页的『SQLError — 检索错误信息』。

DB2 UDB CLI 应用程序的返回码

下表列示了 DB2 UDB CLI 函数的所有可能返回码。第 21 页的第 3 章，『DB2 UDB CLI 函数』中的每个函数描述都列示了每个函数的可能返回码。

表 2. DB2 UDB CLI 函数返回码。

返回码	说明
SQL_SUCCESS	函数成功完成，没有附加的 SQLSTATE 信息可用。
SQL_SUCCESS_WITH_INFO	函数成功完成，但有警告或其它信息。请调用 SQLError() 以接收 SQLSTATE 和任何其它错误信息。SQLSTATE 的类将是 “01”。
SQL_NO_DATA_FOUND	函数成功返回，但找不到相关的数据。
SQL_ERROR	函数失败。请调用 SQLError() 以接收 SQLSTATE 和任何其它错误信息。
SQL_INVALID_HANDLE	由于输入句柄（环境、连接或语句句柄）无效，所以函数失败。

DB2 UDB CLI SQLSTATE

由于不同的数据库服务器通常有不同的诊断消息代码，所以 DB2 UDB CLI 提供了一组标准的由 X/Open SQL CAE 规范定义的 *SQLSTATE*。这样就可以跨不同的数据库服务器进行一致的消息处理。

SQLSTATE 是 5 个字符（字节）的字母数字字符串，其格式为 ccsss，其中 cc 指示类，而 sss 指示子类。任何 SQLSTATE:

- 如果具有“01”类，则是警告。
- 如果具有“HY”类，则是由命令行界面（CLI）驱动程序（DB2 UDB CLI 或“开放式数据库连接”（ODBC））生成的。

如果服务器生成了*本机*错误代码，则 SQLError() 函数还将返回该代码。当连接到 IBM 数据库服务器时，本机错误代码将是 SQLCODE。如果该代码由 DB2 UDB CLI 生成而不是在服务器上生成的，则将把本机错误代码设置为 -99999。

DB2 UDB CLI SQLSTATE 既包括 IBM 定义的由数据库服务器返回的附加 SQLSTATE，也包括 DB2 UDB CLI 定义的用于 X/Open 规范中未定义的情况的 SQLSTATE。这样就可以返回最大量的诊断信息。当使用 ODBC 来在 Windows 中运行应用程序时，接收由 ODBC 定义的 SQLSTATE 也是有可能的。

请遵循下列关于在应用程序中使用 SQLSTATE 的准则:

- 在调用 SQLError() 之前，始终检查函数返回码以确定是否有诊断信息可用。
- 使用 SQLSTATE，而不是使用本机错误代码。
- 为了提高应用程序的可移植性，您只应该将相关性构建在 X/Open 规范定义的 DB2 UDB CLI SQLSTATE 子集之上，并且只将附加的 SQLSTATE 作为信息返回。（相关性是指应用程序根据特定 SQLSTATE 来进行逻辑流决策。）
- 要获得最大量的诊断信息，请将文本消息随 SQLSTATE 一起返回（如果适用的话，文本消息将包含 IBM 定义的 SQLSTATE）。应用程序打印出返回错误的函数的名称也是非常有用的。

DB2 UDB CLI 函数中的数据类型和数据转换

表 3 显示了所有受支持的 SQL 类型和它们的对应符号名。SQLBindParam()、SQLBindParameter()、SQLSetParam()、SQLBindCol() 和 SQLGetData() 使用符号名来指示自变量的数据类型。

下面对每个列作了描述。

SQL 类型

这个列包含出现在 SQL 语句中的 SQL 数据类型。SQL 数据类型取决于 DBMS。

SQL 符号

这个列包含（在 sqlcli.h 中）作为整数值定义的 SQL 符号名。各函数使用这个值来标识第一列中的 SQL 数据类型。

表 3. SQL 数据类型和缺省的 C 数据类型

SQL 类型	SQL 符号
CHAR	SQL_CHAR, SQL_WCHAR[2]
VARCHAR	SQL_VARCHAR, SQL_WVARCHAR[2]
GRAPHIC	SQL_GRAPHIC
VARGRAPHIC	SQL_VARGRAPHIC
SMALLINT	SQL_SMALLINT
BIGINT	SQL_BIGINT
INTEGER	SQL_INTEGER
DECIMAL	SQL_DECIMAL
NUMERIC	SQL_NUMERIC
DOUBLE	SQL_DOUBLE

表 3. SQL 数据类型和缺省的 C 数据类型 (续)

SQL 类型	SQL 符号
FLOAT	SQL_FLOAT
REAL	SQL_REAL
DATE[1]	SQL_CHAR
TIME[1]	SQL_CHAR
TIMESTAMP[1]	SQL_CHAR
BLOB	SQL_BLOB
CLOB	SQL_CLOB
DBCLOB	SQL_DBCLOB
注:	
[1]	DATE、TIME 和 TIMESTAMP 值将以字符格式返回。
[2]	SQL_WCHAR 和 SQL_WVARCHAR 可用来指示 Unicode 数据。

有关更多信息，参见:

- 『DB2 UDB CLI 函数中的其它 C 数据类型』
- 『DB2 UDB CLI 函数中的数据转换』

DB2 UDB CLI 函数中的其它 C 数据类型

除了映射到 SQL 数据类型的数据类型之外，还有一些用于其它函数自变量（如指针和句柄）的 C 符号类型。

表 4. 类属数据类型和实际的 C 数据类型

符号类型	实际的 C 类型	典型用法
SQLPOINTER	void *	指向数据和参数的存储器的指针。
SQLHENV	long int	引用环境信息的句柄。
SQLHDBC	long int	引用数据库连接信息的句柄。
SQLHSTMT	long int	引用语句信息的句柄。
SQLRETURN	long int	DB2 UDB CLI 函数的返回码。

DB2 UDB CLI 函数中的数据转换

就象前面提到的那样，DB2 UDB CLI 管理着数据在应用程序与 DBMS 之间的传送以及任何必需的转换。在发生实际的数据传送之前，在调用 SQLBindParam()、SQLBindParameter()、SQLSetParam()、SQLBindCol() 或 SQLGetData() 时指示源和／或目标数据类型。这些函数使用第 16 页的表 3 中显示的符号类型名来标识所涉及的数据类型。有关使用符号数据类型的函数的示例，请参考 SQLFetch()（第 98 页的『示例』）或 SQLGetCol()（第 119 页的『示例』）。

第 18 页的表 5 显示了 DB2 UDB CLI 支持的转换。只显示了缺省转换。通过使用 SQL 标量函数或在所执行的语句的 SQL 语法中使用 SQL CAST 函数，可以进行其它转换。

可使用前一段中提到的函数将数据转换为其它类型。并非所有数据转换都受支持或有意义。第 18 页的表 5 显示了 DB2 UDB CLI 支持的转换。

第 18 页的表 5 的第一列包含源的数据类型，其余各列表示目标数据类型。X 表示 DB2 UDB CLI 支持该转换。

表 5. 支持的数据转换

源数据类型	VARGRAPHIC	GRAPHIC	TIMESTAMP	TIME	DATE	VARCHAR	DOUBLE	REAL	FLOAT	SMALLINT	BIGINT	INTEGER	DECIMAL	NUMERIC	CHAR	BLOB	CLOB	DBCLOB
CHAR VARCHAR			X	X	X	X				X					X		X	
GRAPHIC VARGRAPHIC	X	X																X
BLOB																X		
CLOB						X									X		X	
DBCLOB	X	X																X
INTEGER SMALLINT BIGINT DECIMAL NUMERIC DOUBLE FLOAT						X	X	1	X	X	X	X	X	2	X			
DATE			X		X	X									X			
TIME			X	X		X									X			
TIMESTAMP			X	X	X	X									X			

注:

1. DB2 UDB OS/2® 版或 DB2 UDB AIX/6000 版不支持 REAL
2. 只有 DB2 UDB iSeries 版才支持 NUMERIC，其它 DBMS 将其视为 DECIMAL

每当函数调用上发生取整截断或数据类型不兼容时，将返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO。于是，进一步的信息由 SQLSTATE 值和 SQLError() 返回的其它信息指示。

在 DB2 UDB CLI 函数中使用字符串自变量

下列约定涉及与在 DB2 UDB CLI 函数中使用字符串自变量相关的各个方面。

- 第 19 页的『DB2 UDB CLI 函数中的字符串自变量的长度』
- 第 19 页的『DB2 UDB CLI 函数中的字符串截断』
- 第 19 页的『DB2 UDB CLI 函数中的字符串的解释』

DB2 UDB CLI 函数中的字符串自变量的长度

输入字符串自变量具有相关联的长度自变量。这个自变量向 DB2 UDB CLI 指示已分配的缓冲区的长度（不包括空字节终止符）或特殊值 SQL_NTS。如果传送 SQL_NTS，则 DB2 UDB CLI 通过定位空终止字符来确定字符串的长度。

输出字符串自变量具有两个相关联的长度自变量，一个用于指定已分配的缓冲区的长度，一个用于返回由 DB2 UDB CLI 返回的字符串的长度。返回的长度值是可供返回的字符串的总长度，而无论该字符串在缓冲区中是否放得下。

对于 SQL 列数据，如果输出是空字符串，则在长度自变量中返回 SQL_NULL_DATA。

如果通过对输出长度自变量使用空指针来调用函数，则 DB2 UDB CLI 不返回长度。当已经知道缓冲区对于所有可能的结果而言都足够大时，这可能很有用。如果 DB2 UDB CLI 尝试返回 SQL_NULL_DATA 值以指示列包含空数据，并且输出长度自变量是空指针，则函数调用将失败。

除了从图形数据类型返回的字符串之外，DB2 UDB CLI 返回的每个字符串都以空终止字符（十六进制 00）终止。这要求所有缓冲区为所期望的最大字符数分配足够的空间，另外再分配一个字节供空终止字符使用。

DB2 UDB CLI 函数中的字符串截断

如果输出字符串在缓冲区中装不下，则 DB2 UDB CLI 将截断字符串，使其长度比缓冲区的大小小 1，并写空终止符。如果发生截断，则函数返回 SQL_SUCCESS_WITH_INFO 和 SQLSTATE 指示截断。然后，应用程序可将缓冲区长度与输出长度作比较以确定截断了哪个字符串。

例如，如果 `SQLFetch()` 返回 SQL_SUCCESS_WITH_INFO 以及 01004 的 SQLSTATE，则表示至少一个绑定到列的缓冲区太小，无法存放数据。对于每个绑定到列的缓冲区，应用程序可以将缓冲区长度与输出长度作比较并确定截断了哪个字符串。

DB2 UDB CLI 函数中的字符串的解释

对于所有字符串输入自变量，如列名和游标名，DB2 UDB CLI 忽略大小写并除去前导和尾部空格，但对于下列各项例外:

- 任何数据库数据
- 括在双引号中的定界标识符
- 密码自变量

第 3 章 DB2 UDB CLI 函数

本章提供每个函数的描述。每个 DB2 UDB CLI 函数都包含下列信息:

- **用途**

  此部分给出函数功能的简要概述。此部分还指示了在调用所描述的函数之前和之后是否应调用任何函数。

- **语法**

  此部分包含用于 OS/400® 环境的 “C” 原型。

- **自变量**

  此部分列示每个函数自变量及其数据类型、描述以及它是输入还是输出自变量。

  每个 DB2 UDB CLI 自变量都是输入或输出自变量。除 `SQLGetInfo()` 之外，DB2 UDB CLI 只修改作为输出指示的自变量。

  一些函数包含输入或输出自变量，这些自变量称为*延迟*或*绑定*自变量。这些自变量是指向应用程序分配的缓冲区的指针。这些自变量与 SQL 语句中的参数或结果集中的列相关联（或绑定到该参数或列）。以后，DB2 UDB CLI 访问由函数指定的数据区。当 DB2 UDB CLI 访问这些延迟数据区的时候它们仍有效这一点至关重要。

- **用法**

  此部分提供关于如何使用该函数的信息以及任何特殊的注意事项。这里没有讨论可能的错误情况，而是将它们列示在诊断部分中。

- **返回码**

  此部分列示所有可能的函数返回码。当返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，可通过调用 `SQLError()` 来获取错误信息。

  关于返回码的更多信息，请参考第 15 页的『DB2 UDB CLI 应用程序中的诊断』。

- **诊断**

  此部分包含一个表，这个表列示了 DB2 UDB CLI 显式返回的 SQLSTATE（还可能返回 DBMS 生成的 SQLSTATE）并指示了错误的原因。这些值是在函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 之后通过调用 `SQLError()` 获取的。

  第一列中的 ***** 指示只有 DB2 UDB CLI 返回该 SQLSTATE，其它 ODBC 驱动程序不返回它。

  有关诊断的更多信息，请参考第 15 页的『DB2 UDB CLI 应用程序中的诊断』。

- **限制**

  此部分指示 DB2 UDB CLI 与 ODBC 之间的可能影响应用程序的任何差别或限制。

- **示例**

  此部分是演示函数用法的代码段。第 261 页的附录 C，『示例 DB2 UDB CLI 应用程序代码列表』列示了用于所有代码段的完整源代码。

- **参考**

  此部分列示相关的 DB2 UDB CLI 函数。

函数如下:

SQLAllocConnect — 分配连接句柄

用途

SQLAllocConnect() 在由输入环境句柄标识的环境中分配连接句柄和相关联的资源。在将 fInfoType 设置为 SQL_ACTIVE_CONNECTIONS 的情况下调用 SQLGetInfo() 以查询在任何一个时候可以分配的连接数。

在调用此函数之前，必须调用 SQLAllocEnv()。

语法

```
SQLRETURN SQLAllocConnect (SQLHENV    henv,
                           SQLHDBC    *phdbc);
```

函数自变量

表 6. SQLAllocConnect 自变量

数据类型	自变量	用途	描述
SQLHENV	*henv*	输入	环境句柄
SQLHDBC *	*phdbc*	输出	指向连接句柄的指针

用法

DB2 UDB CLI 使用输出连接句柄来引用与连接相关的所有信息，包括一般状态信息、事务状态和错误信息。

如果指向连接句柄的指针（*phdbc*）指向由 SQLAllocConnect() 分配的有效连接句柄，则此调用的结果是覆盖原始值。这是应用程序编程错误，DB2 UDB CLI 不检测此错误。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

如果返回 SQL_ERROR，则 *phdbc* 自变量设置为 SQL_NULL_HDBC。应用程序应使用环境句柄（*henv*）来调用 SQLError()，并且将 *hdbc* 和 *hstmt* 自变量分别设置为 SQL_NULL_HDBC 和 SQL_NULL_HSTMT。

诊断

表 7. SQLAllocConnect SQLSTATE

CLI SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	*phdbc* 是空指针

示例

以下示例显示如何获取连接和环境的诊断信息。有关更多使用 SQLError() 的示例，请参考第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』以获取 typical.c 的完整列表。

有关代码示例的信息，参见第 viii 页的『代码不保证声明信息』。

```
/*************************************************************
** initialize
**  - allocate environment handle
**  - allocate connection handle
**  - prompt for server, user id, & password
**  - connect to server
*************************************************************/

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc)
{
SQLCHAR     server[SQL_MAX_DSN_LENGTH],
            uid[30],
            pwd[30];
SQLRETURN   rc;

    SQLAllocEnv (henv);         /* allocate an environment handle   */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    SQLAllocConnect (*henv, hdbc);  /* allocate a connection handle     */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
    else
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
}/* end initialize */


/*************************************************************/
int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
        else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
```

```
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);

}
```

参考

- 第 27 页的『SQLAllocEnv — 分配环境句柄』
- 第 68 页的『SQLConnect — 连接到数据源』
- 第 80 页的『SQLDisconnect — 与数据源断开连接』
- 第 110 页的『SQLFreeConnect — 释放连接句柄』
- 第 121 页的『SQLGetConnectAttr — 获取连接属性的值』
- 第 210 页的『SQLSetConnectOption — 设置连接选项』

SQLAllocEnv — 分配环境句柄

用途

SQLAllocEnv() 分配环境句柄和相关联的资源。

应用程序必须在 SQLAllocConnect() 或任何其它 DB2 UDB CLI 函数之前调用此函数。在以后所有需要环境句柄作为输入的函数调用中，都传送 *henv* 值。

语法

```
SQLRETURN SQLAllocEnv (SQLHENV    *phenv);
```

函数自变量

表 8. SQLAllocEnv 自变量

数据类型	自变量	用途	描述
SQLHENV *	*phenv*	输出	指向环境句柄的指针

用法

对于每个应用程序，在任何一个时候都只能有一个活动环境。以后任何对 SQLAllocEnv() 的调用将返回现有的环境句柄。

缺省情况下，对 SQLFreeEnv() 所作的第一个成功调用将释放与该句柄相关联的资源。无论成功地调用了多少次 SQLAllocEnv()，都会发生这种情况。如果环境属性 SQL_ATTR_ENVHNDL_COUNTER 设置为 SQL_TRUE，则在释放与句柄相关联的资源之前，必须为每次成功的 SQLAllocEnv() 调用来调用 SQLFreeEnv()。

要确保所有的 DB2 UDB CLI 资源都保持活动状态，调用 SQLAllocEnv() 的程序不应终止或离开堆栈。否则，应用程序将丢失打开的游标、语句句柄和其它已分配的资源。

返回码

- SQL_SUCCESS
- SQL_ERROR

如果返回 SQL_ERROR，并且 *phenv* 等于 SQL_NULL_HENV，则由于没有句柄可以与附加的诊断信息相关联，所以不能调用 SQLError()。

如果返回码是 SQL_ERROR，并且指向环境句柄的指针不等于 SQL_NULL_HENV，则该句柄是**受限句柄**。这表示该句柄只能在 SQLError() 的调用中使用以获取更多的错误信息，或者用于 SQLFreeEnv() 的调用。

诊断

表 9. SQLAllocEnv SQLSTATE

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误。

示例

有关代码示例的信息，参见第 viii 页的『代码不保证声明信息』。

```
/*************************************************
** file = basiccon.c
**    - demonstrate basic connection to two datasources.
**    - error handling  ignored for simplicity
**
**  Functions used:
**
**    SQLAllocConnect  SQLDisconnect
**    SQLAllocEnv      SQLFreeConnect
**    SQLConnect       SQLFreeEnv
**
**
*************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "sqlcli.h"

int
connect(SQLHENV henv,
        SQLHDBC * hdbc);

#define MAX_DSN_LENGTH    18
#define MAX_UID_LENGTH    10
#define MAX_PWD_LENGTH    10
#define MAX_CONNECTIONS   5

int
main()
{
    SQLHENV         henv;
    SQLHDBC         hdbc[MAX_CONNECTIONS];

    /* allocate an environment handle   */
    SQLAllocEnv(&henv);

    /* Connect to first data source */
    connect(henv, &hdbc[0];);

    /* Connect to second data source */
    connect(henv, &hdbc[1];);

    /*********   Start Processing Step  ************************/
    /* allocate statement handle, execute statement, etc.       */
    /*********   End Processing Step  *************************/

    printf("\nDisconnecting .....\n");
    SQLFreeConnect(hdbc[0]);    /* free first connection handle  */
    SQLFreeConnect(hdbc[1]);    /* free second connection handle */
    SQLFreeEnv(henv);           /* free environment handle       */

    return (SQL_SUCCESS);
}

/*********************************************************************
**   connect - Prompt for connect options and connect              **
*********************************************************************/

int
connect(SQLHENV henv,
        SQLHDBC * hdbc)
{
    SQLRETURN       rc;
```

```
    SQLCHAR         server[MAX_DSN_LENGTH + 1], uid[MAX_UID_LENGTH + 1],
pwd[MAX_PWD_LENGTH
+ 1];
    SQLCHAR         buffer[255];
    SQLSMALLINT     outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    SQLAllocConnect(henv, hdbc);/* allocate a connection handle     */

    rc = SQLConnect(*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database\n");
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
        return (SQL_SUCCESS);
    }
}
```

参考

- 第 24 页的『SQLAllocConnect — 分配连接句柄』
- 第 111 页的『SQLFreeEnv — 释放环境句柄』
- 第 32 页的『SQLAllocStmt — 分配语句句柄』

SQLAllocHandle — 分配句柄

用途

SQLAllocHandle() 分配任何类型的句柄。

语法

```
SQLRETURN SQLAllocHandle (SQLSMALLINT htype,
                          SQLINTEGER ihandle,
                          SQLINTEGER *handle);
```

函数自变量

表 10. SQLAllocHandle 自变量

数据类型	自变量	用途	描述
SQLSMALLINT	*htype*	输入	要分配的句柄的类型。必须是 SQL_HANDLE_ENV、SQL_HANDLE_DBC、SQL_HANDLE_DESC 或 SQL_HANDLE_STMT。
SQLINTEGER	*ihandle*	输入	描述在其中分配新句柄的上下文的句柄；然而，如果 *htype* 是 SQL_HANDLE_ENV，则这是 SQL_NULL_HANDLE。
SQLINTEGER *	*handle*	输出	指向句柄的指针

用法

此函数组合了函数 SQLAllocEnv()、SQLAllocConnect() 和 SQLAllocStmt()。

如果 *htype* 是 SQL_HANDLE_ENV，则 *ihandle* 必须是 SQL_NULL_HANDLE。如果 *htype* 是 SQL_HANDLE_DBC，则 *ihandle* 必须是有效的环境句柄。如果 *htype* 是 SQL_HANDLE_DESC 或 SQL_HANDLE_STMT，则 *ihandle* 必须是有效的连接句柄。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

如果自变量 handle 是空指针，则返回 SQL_ERROR。

表 11. SQLAllocHandle SQLSTATE

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误。
HY014	句柄太多	已分配了最大数目的句柄

参考

- 第 24 页的『SQLAllocConnect — 分配连接句柄』
- 第 27 页的『SQLAllocEnv — 分配环境句柄』
- 第 32 页的『SQLAllocStmt — 分配语句句柄』

SQLAllocStmt — 分配语句句柄

用途

SQLAllocStmt() 分配新的语句句柄并将其与连接句柄指定的连接相关联。对于在任何一个时候可以分配的语句句柄数目没有已定义的限制。

在调用此函数之前，必须调用 SQLConnect()。

必须在 SQLBindParam()、SQLPrepare()、SQLExecute()、SQLExecDirect() 或任何其它将语句句柄作为其中一个输入自变量的函数之前调用此函数。

语法

```
SQLRETURN SQLAllocStmt (SQLHDBC    hdbc,
                        SQLHSTMT   *phstmt);
```

函数自变量

表 12. SQLAllocStmt 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄
SQLHSTMT *	*phstmt*	输出	指向语句句柄的指针

用法

DB2 UDB CLI 使用每个语句句柄来使所有描述符、结果值、游标信息和状态信息与已处理的 SQL 语句相关联。虽然每个 SQL 语句都必须要有语句句柄，但可以将句柄重新用于不同的语句。

对此函数的调用要求 *hdbc* 引用活动的数据库连接。

要执行定位型更新或删除，应用程序必须将不同的语句句柄用于 SELECT 语句和 UPDATE 或 DELETE 语句。

如果指向语句句柄的输入指针（*phstmt*）指向由先前的 SQLAllocStmt() 调用分配的有效语句句柄，则此调用的结果是覆盖原始值。这是应用程序编程错误，DB2 UDB CLI 不检测此错误。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

如果返回 SQL_ERROR，则 *phstmt* 自变量设置为 SQL_NULL_HSTMT。应用程序应使用同一个 *hdbc* 来调用 SQLError() 并将 *hstmt* 自变量设置为 SQL_NULL_HSTMT。

诊断

表 13. SQLAllocStmt SQLSTATE

SQLSTATE	描述	说明
08003	连接未打开	*hdbc* 自变量指定的连接尚未打开。必须成功地为驱动程序建立连接（并且该连接必须已打开）才能分配 *hstmt*。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	*phstmt* 是空指针
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

请参考 `SQLFetch()`（第 98 页的『示例』）。

参考

- 第 68 页的『SQLConnect — 连接到数据源』
- 第 114 页的『SQLFreeStmt — 释放（或重设）语句句柄』
- 第 162 页的『SQLGetStmtOption — 返回语句选项的当前设置』
- 第 226 页的『SQLSetStmtOption — 设置语句选项』

SQLBindCol — 将列绑定到应用程序变量

用途

SQLBindCol() 将结果集中的列与应用程序变量（存储器缓冲区）相关联（绑定），这适用于所有数据类型。调用 SQLFetch() 时，将把数据从 DBMS 传送到应用程序。

此函数还用来指定所需的任何数据转换。对应用程序在结果集中需要检索的每一列调用此函数一次。

通常，在此函数之前调用 SQLPrepare() 或 SQLExecDirect()。可能还有必要调用 SQLDescribeCol() 或 SQLColAttributes()。

在调用 SQLFetch() 之前，必须调用 SQLBindCol() 以将数据传送至此调用指定的存储器缓冲区。

语法

```
SQLRETURN SQLBindCol (SQLHSTMT       hstmt,
                      SQLSMALLINT    icol,
                      SQLSMALLINT    fCType,
                      SQLPOINTER     rgbValue,
                      SQLINTEGER     cbValueMax,
                      SQLINTEGER     *pcbValue);
```

函数自变量

表 14. SQLBindCol 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*icol*	输入	用于标识列的编号。列是从 1 开始从左到右按顺序编号的。

表 14. SQLBindCol 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT	*fCType*	输入	结果集中的编号为 *icol* 的列的应用程序数据类型。支持下列类型: • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR 指定 SQL_DEFAULT 将导致将数据转换为它的缺省数据类型，有关更多信息，请参考第 16 页的表 3。
SQLPOINTER	*rgbValue*	输出（延迟）	指向执行取装操作时 DB2 UDB CLI 将用来存储列数据的缓冲区的指针。 如果 *rgbValue* 为空，则将列取消绑定。

表 14. SQLBindCol 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER	*cbValueMax*	输入	可用来存储列数据的 *rgbValue* 缓冲区的大小（字节）。 如果 *fCType* 是 SQL_CHAR 或 SQL_DEFAULT，则 *cbValueMax* 必须大于 0，否则将返回错误。 如果 *fcType* 是 SQL_DECIMAL 或 SQL_NUMERIC，则 *cbValueMax* 必须实际上是精度和标度。指定这两个值的方法是使用 *(精度 * 256) + 标度*。这也是使用 SQLColAttributes() 时作为这些数据类型的长度（LENGTH）返回的值。 如果 *fcType* 指定任何形式的双字节字符数据，则 *cbValueMax* 必须是双字节字符数而不是字节数。
SQLINTEGER *	*pcbValue*	输出（延迟）	指向一个值的指针，该值指示了 DB2 UDB CLI 可以在 *rgbValue* 缓冲区中返回的字节数。 如果列的数据值为空，则 SQLFetch() 在此自变量中返回 SQL_NULL_DATA。如果列的数据值是作为以空终止的字符串返回的，则在此自变量中返回 SQL_NTS。

注:

对于此函数，*rgbValue* 和 *pcbValue* 都是延迟输出，这表示在调用 SQLFetch() 之前不更新这些指针指向的存储器位置。在调用 SQLFetch() 之前，这些指针引用的位置必须保持有效。

用法

应用程序对于它在结果集中要检索的每一列调用一次 SQLBindCol()。在调用 SQLFetch() 时，将把每个*绑定*列中的数据放到指定的位置（由指针 *rgbValue* 和 *pcbValue* 给出）中。

应用程序可以通过首先调用 SQLDescribeCol() 或 SQLColAttributes() 来查询列的属性（如数据类型和长度）。然后，可使用此信息来指定存储器位置的正确数据类型，或指示将数据转换为其它数据类型。有关更多信息，请参考第 16 页的『DB2 UDB CLI 函数中的数据类型和数据转换』。

在以后的取装操作中，应用程序可以更改这些列的绑定，或者通过调用 SQLBindCol() 来绑定未绑定的列。新的绑定不会应用于已取装的数据，而是在下次调用 SQLFetch() 时使用。要将单个列取消绑定，请调用 SQLBindCol()，并将 *rgbValue* 设置为空。要将所有列取消绑定，应用程序应调用 SQLFreeStmt()，并将 *fOption* 输入设置为 SQL_UNBIND。

列由编号标识，这个编号是从 1 开始从左到右按顺序指定的。可通过调用 SQLNumResultCols() 或 SQLColAttributes() 并将 *fdescType* 自变量设置为 SQL_DESC_COUNT 来确定结果集中的列数。

应用程序可以选择不绑定每个列，甚至不绑定任何列。在调用 SQLFetch() 之后，可使用 SQLGetData() 来检索未绑定列（并且只是未绑定列）中的数据。SQLBindCol() 的效率比 SQLGetData() 高，应该尽可能使用它。

应用程序必须确保为将要检索的数据分配足够的存储器。如果缓冲区将要包含变长数据，则应用程序必须根据绑定列所需的最大长度来分配存储器，否则数据可能会被截断。

如果发生字符串截断，则会返回 SQL_SUCCESS_WITH_INFO，并且将 *pcbValue* 设置为可以返回给应用程序的 *rgbValue* 的实际大小。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 15. SQLBindCol SQLSTATE

SQLSTATE	描述	说明
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY002	列号无效	对自变量 *icol* 指定的值是 0。 对自变量 *icol* 指定的值超出数据源所支持的最大列数。
HY003	程序类型超出范围	*fCType* 不是有效的数据类型
HY009	自变量值无效	*rgbValue* 是空指针 对自变量 *cbValueMax* 指定的值小于 1，并且自变量 *fCType* 是 SQL_CHAR 或 SQL_DEFAULT。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。
HY014	句柄太多	已分配了最大数目的句柄，使用此函数将需要附加的描述符句柄。
HYC00	驱动程序不具有能力	驱动程序能够识别但不支持自变量 *fCType* 中指定的驱动程序（另见 **HY**003）。

示例

请参考 `SQLFetch()`（第 98 页的『示例』）。

参考

- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 186 页的『SQLPrepare — 准备语句』

SQLBindFileToCol — 将 LOB 文件引用绑定到 LOB 列

用途

SQLBindFileToCol() 用来将结果集中的 LOB 列与文件引用或文件引用数组相关联（绑定）。这样，在为语句句柄取装每一行时，能够将该列中的数据直接传送到文件中。

LOB 文件引用自变量（文件名、文件名长度和文件引用选项）引用（客户机上）应用程序的环境内的文件。在取装每一行之前，应用程序必须确保这些变量包含文件的名称、文件名的长度和文件选项（新建／覆盖／附加）。可以在两次取装操作之间更改这些值。

语法

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

函数自变量

表 16. SQLBindFileToCol 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLSMALLINT	*ColumnNumber*	输入	用于标识列的编号。列是从 1 开始从左到右按顺序编号的。
SQLCHAR *	*FileName*	输入（延迟）	指向一个位置的指针，在使用 *StatementHandle* 进行下次取装时，该位置将包含文件名或文件名数组。这是文件的完整路径名，或者是相对文件名。如果提供相对文件名，则把它们附加到正在运行的应用程序的当前路径。此指针不能为空。
SQLSMALLINT *	*FileNameLength*	输入（延迟）	指向一个位置的指针，在使用 *StatementHandle* 进行下次取装时，该位置将包含文件名的长度（或长度数组）。如果此指针为空，则假定长度为 SQL_NTS。 文件名长度的最大值是 255。

表 16. SQLBindFileToCol 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER *	*FileOptions*	输入（延迟）	指向一个位置的指针，在使用 *StatementHandle* 进行下次取装时，该位置将包含写文件时要使用的文件选项。支持下列 *FileOptions*: **SQL_FILE_CREATE** 创建新文件。如果已存在具有此名称的文件，则将返回 SQL_ERROR。 **SQL_FILE_OVERWRITE** 如果文件已存在，则将其覆盖。否则，创建新文件。 **SQL_FILE_APPEND** 如果文件已存在，则将数据附加到该文件。否则，创建新文件。 对每个文件只能选择一个选项，没有缺省值。
SQLSMALLINT	*MaxFileNameLength*	输入	此自变量指定 *FileName* 缓冲区的长度。
SQLINTEGER *	*StringLength*	输出（延迟）	指向一个位置的指针，该位置包含所返回的 LOB 数据的长度（字节）。如果此指针为空，则表示未返回任何内容。
SQLINTEGER *	*IndicatorValue*	输出（延迟）	指向包含指示符值的位置的指针。

用法

对于每一个在取装行时应当直接传送到文件中的列，应用程序调用一次 `SQLBindFileToCol()`。将把 LOB 数据直接写至文件，不进行任何数据转换，并且不附加空终止符。

在每次取装之前，必须设置 *FileName*、*FileNameLength* 和 *FileOptions*。在调用 `SQLFetch()` 或 `SQLFetchScroll()` 时，将把任何已绑定到 LOB 文件引用的列的数据写至由该文件引用指向的一个或多个文件。在取装时将报告与 `SQLBindFileToCol()` 的延迟输入自变量值相关联的错误。在两次取装操作之间更新 LOB 文件引用以及 *StringLength* 和 *IndicatorValue* 延迟输出自变量。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 17. SQLBindFileToCol SQLSTATE

SQLSTATE	描述	说明
58004	意外的系统故障	不可恢复的系统错误。
HY002	列号无效	对自变量 *icol* 指定的值小于 1。 对自变量 *icol* 指定的值超出数据源所支持的最大列数。
HY009	自变量值无效	*FileName*、*StringLength* 或 *FileOptions* 是空指针。

表 17. SQLBindFileToCol SQLSTATE (续)

SQLSTATE	描述	说明
HY010	函数顺序错误	在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。 在执行 BEGIN COMPOUND 和 END COMPOUND SQL 操作时调用了此函数。
HY090	字符串或缓冲区长度无效	对自变量 *MaxFileNameLength* 指定的值小于 0。
HYC00	驱动程序不具有能力	应用程序当前连接到不支持大对象的数据源。

限制

当连接到不支持“大对象”数据类型的 DB2 服务器时，此函数不可用。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 41 页的『SQLBindFileToParam — 将 LOB 文件引用绑定到 LOB 参数』

SQLBindFileToParam — 将 LOB 文件引用绑定到 LOB 参数

用途

SQLBindFileToParam() 用来使 SQL 语句中的参数标记与文件引用或文件引用数组相关联（绑定）。这使得以后执行该语句时可以将该文件中的数据直接传送到 LOB 列中。

LOB 文件引用自变量（文件名、文件名长度和文件引用选项）引用（客户机上）应用程序的环境内的文件。在调用 SQLExecute() 或 SQLExecDirect() 之前，应用程序必须确保此信息已位于延迟输入缓冲区中。可以在两次 SQLExecute() 调用之间更改这些值。

语法

```
SQLRETURN SQLBindFileToParam (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       ParameterNumber,
                              SQLSMALLINT       DataType,
                              SQLCHAR           *FileName,
                              SQLSMALLINT       *FileNameLength,
                              SQLINTEGER        *FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER        *IndicatorValue);
```

函数自变量

表 18. SQLBindFileToParam 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLSMALLINT	*ParameterNumber*	输入	参数标记编号。参数是从 1 开始从左到右按顺序编号的。
SQLSMALLINT	*DataType*	输入	列的 SQL 数据类型。数据类型必须是下列其中之一： • SQL_BLOB • SQL_CLOB • SQL_DBCLOB
SQLCHAR *	*FileName*	输入（延迟）	指向一个位置的指针，在执行语句（*StatementHandle*）时，该位置将包含文件名或文件名数组。这是文件的完整路径名，或者是相对文件名。如果提供相对文件名，则将把它附加到客户机进程的当前路径。 此自变量不能为空。
SQLSMALLINT *	*FileNameLength*	输入（延迟）	指向一个位置的指针，在使用 *StatementHandle* 进行下次 SQLExecute() 或 SQLExecDirect() 调用时，该位置将包含文件名的长度（或长度数组）。 如果此指针为空，则假定长度为 SQL_NTS。 文件名长度的最大值是 255。

表 18. SQLBindFileToParam 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER *	*FileOptions*	输入（延迟）	指向一个位置的指针，该位置将包含在读取文件时要使用的文件选项（或文件选项数组）。在执行语句（*StatementHandle*）时，将访问该位置。只支持一个选项（并且必须指定）： **SQL_FILE_READ** 可以打开、读取和关闭的规则文件。（在打开文件时计算长度） 此指针不能为空。
SQLSMALLINT	*MaxFileNameLength*	输入	此自变量指定 *FileName* 缓冲区的长度。如果应用程序调用 SQLParamOptions() 来对每个参数指定多个值，则这是 *FileName* 数组中的每个元素的长度。
SQLINTEGER *	*IndicatorValue*	输出（延迟）	指向包含指示符值（或值数组）的位置的指针，如果参数的数据值将为空，则该指示符设置为 SQL_NULL_DATA。当数据值非空时，必须将该指示符设置为 0（也可以将指针设置为空）。

用法

对于每个在执行语句时应当直接从文件中获取值的参数标记，应用程序调用一次 SQLBindFileToParam()。在执行语句之前，必须设置 *FileName*、*FileNameLength* 和 *FileOptions* 值。执行语句时，将从所引用的文件中读取任何已使用 SQLBindFIleToParam() 绑定了的参数的数据并将该数据传送给服务器。

可使用 SQLBindFileToParam() 来使 LOB 参数标记与输入文件相关联（绑定），或使用 SQLBindParameter() 来使其与存储器缓冲区相关联。最近的绑定参数函数调用确定了起作用的绑定的类型。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 19. SQLBindFileToParam SQLSTATE

SQLSTATE	描述	说明
58004	意外的系统故障	不可恢复的系统错误。
HY004	SQL 数据类型超出范围	对 *DataType* 指定的值对此函数调用而言不具有有效的 SQL 类型。
HY009	自变量值无效	*FileName*、*FileOptions* 或 *FileNameLength* 是空指针。
HY010	函数顺序错误	在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。 在执行 BEGIN COMPOUND 和 END COMPOUND SQL 操作时调用了此函数。
HY090	字符串或缓冲区长度无效	对输入自变量 *MaxFileNameLength* 指定的值小于 0。

表 19. SQLBindFileToParam SQLSTATE (续)

SQLSTATE	描述	说明
HY093	参数编号无效	对 *ParameterNumber* 指定的值小于 1 或大于所支持的最大参数数目。
HYC00	驱动程序不具有能力	服务器不支持“大对象”数据类型。

限制

当连接到不支持“大对象”数据类型的 DB2 服务器时，此函数不可用。

参考

- 第 44 页的『SQLBindParam — 将缓冲区绑定到参数标记』
- 第 93 页的『SQLExecute — 执行语句』
- 第 184 页的『SQLParamOptions — 指定参数的输入数组』

SQLBindParam — 将缓冲区绑定到参数标记

用途

SQLBindParam() 将应用程序变量绑定到 SQL 语句中的参数标记。此函数还可用来将应用程序变量绑定到存储过程 CALL 语句的参数，该参数可以是输入参数，也可以是输出参数。此函数与 SQLSetParam() 相同。

语法

```
SQLRETURN SQLBindParam (SQLHSTMT    hstmt,
                        SQLSMALLINT ipar,
                        SQLSMALLINT fCType,
                        SQLSMALLINT fSqlType,
                        SQLINTEGER  cbParamDef,
                        SQLSMALLINT ibScale,
                        SQLPOINTER  rgbValue,
                        SQLINTEGER  *pcbValue);
```

函数自变量

表 20. SQLBindParam 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*ipar*	输入	参数标记编号，此编号从 1 开始从左到右按顺序排序。

表 20. SQLBindParam 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT	*fCType*	输入	参数的应用程序数据类型。支持下列类型: • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR 指定 SQL_DEFAULT 将导致将数据由其缺省应用程序数据类型转换为 *fSqlType* 指示的类型。

SQLBindParam

表 20. SQLBindParam 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT	*fSqlType*	输入	参数的 SQL 数据类型。支持的类型包括: • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR
SQLINTEGER	*cbParamDef*	输入	对应的参数标记的精度。如果 *fSqlType* 指示: • 单字节字符串（例如 SQL_CHAR），则此自变量是为此参数发送的最大长度（字节）。此长度包括空终止字符。 • 双字节字符串（例如 SQL_GRAPHIC），则此自变量是此参数的最大长度（以双字节字符数计）。 • SQL_DECIMAL 或 SQL_NUMERIC，则此自变量是最大小数精度。 • 否则，不使用此自变量。
SQLSMALLINT	*ibScale*	输入	相应参数的标度（如果 *fSqlType* 是 SQL_DECIMAL 或 SQL_NUMERIC 的话）。如果 *fSqlType* 是 SQL_TIMESTAMP，则这是时间戳记的字符表示法中小数点右边的位数（例如，yyyy-mm-dd hh:mm:ss.fff 的标度是 3）。 除了用于此处提到的 *fSqlType* 值之外，不使用 *ibScale*。

表 20. SQLBindParam 自变量 (续)

数据类型	自变量	用途	描述
SQLPOINTER	*rgbValue*	输入（延迟） 或 输出（延迟）	在执行时，如果 *pcbValue* 不包含 SQL_NULL_DATA 或 SQL_DATA_AT_EXEC，则 *rgbValue* 指向包含参数的实际数据的缓冲区。 如果 *pcbValue* 包含 SQL_DATA_AT_EXEC，则 *rgbValue* 是应用程序定义的与此参数相关联的 32 位值。这个 32 位值通过以后的 SQLParamData() 调用返回给应用程序。
SQLINTEGER *	*pcbValue*	输入（延迟）、输出（延迟）或输入输出（延迟）	一个变量，在执行语句时将解释这个变量的值: • 如果使用空值作为参数，则 *pcbValue* 必须包含 SQL_NULL_DATA 值。 • 在执行时，如果通过调用 ParamData() 和 PutData() 来提供动态自变量，则 *pcbValue* 必须包含 SQL_DATA_AT_EXEC 值。 • 如果 *fcType* 是 SQL_CHAR，并且 *rgbValue* 中的数据包含以空终止的字符串，则 *pcbValue* 必须包含 *rgbValue* 中的数据的长度或包含 SQL_NTS 值。 • 如果 *fcType* 是 SQL_CHAR，并且 *rgbValue* 中的数据不是以空终止的，则 *pcbValue* 必须包含 *rgbValue* 中的数据的长度。 • 如果 *fcType* 具有 LOB 类型，则 *pcbValue* 必须包含 *rgbValue* 中的数据的长度。 • 否则，*pcbValue* 必须是零。

用法

当使用 SQLBindParam() 来将应用程序变量绑定到存储过程的输出参数时，如果在内存中将 *rgbValue* 缓冲区相邻地放在 *pcbValue* 缓冲区之后，则 DB2 UDB CLI 能提供一定程度的性能增强。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 21. SQLBindParam SQLSTATE

SQLSTATE	描述	说明
07006	受限数据类型属性违例	与 SQLSetParam() 相同。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。

SQLBindParam

表 21. SQLBindParam SQLSTATE (续)

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY003	程序类型超出范围	与 SQLSetParam() 相同。
HY004	SQL 数据类型超出范围	与 SQLSetParam() 相同。
HY009	自变量值无效	*rgbValue 和 pcbValue* 都是空指针，或者 *ipar* 小于 1。
HY010	函数顺序错误	在 SQLExecute() 或 SQLExecDirect() 返回 SQL_NEED_DATA 后调用了此函数，但还没有为所有*执行数据*参数发送数据。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。
HY014	句柄太多	已分配了最大数目的句柄

SQLBindParameter — 将参数标记绑定到缓冲区

用途

SQLBindParameter() 用来使 SQL 语句中的参数标记与应用程序变量相关联（绑定）。当调用 SQLExecute() 或 SQLExecDirect() 时，将把数据从应用程序传送至 DBMS。在传送数据时，可能会发生数据转换。

还必须使用此函数来将应用程序存储器绑定到存储过程 CALL 语句的参数，该参数可以是输入参数、输出参数或输入输出参数。此函数本质上是 SQLSetParam() 的扩展。

语法

```
SQLRETURN SQLBindParameter(SQLHSTMT          StatementHandle,
                           SQLSMALLINT      ParameterNumber,
                           SQLSMALLINT       InputOutputType,
                           SQLSMALLINT       ValueType,
                           SQLSMALLINT       ParameterType,
                           SQLINTEGER        ColumnSize,
                           SQLSMALLINT       DecimalDigits,
                           SQLPOINTER        ParameterValuePtr,
                           SQLINTEGER        BufferLength,
                           SQLINTEGER       *StrLen_or_IndPtr);
```

函数自变量

表 22. SQLBindParameter 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄
SQLSMALLINT	ParameterNumber	输入	参数标记编号，此编号从 1 开始从左到右按顺序排序。

SQLBindParameter

表 22. SQLBindParameter 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT	InputOutputType	输入	参数的类型。IPD 的 SQL_DESC_PARAMETER_TYPE 字段的值也设置为此自变量。支持的类型包括: • SQL_PARAM_INPUT: 参数标记与不是存储过程 CALL 的 SQL 语句相关联; 或者, 它标记调用（CALL）的存储过程的输入参数。 执行语句时, 将把参数的实际数据值发送至服务器: *ParameterValuePtr* 缓冲区必须包含有效的输入数据值; *StrLen_or_IndPtr* 缓冲区必须包含 SQL_NTS、SQL_NULL_DATA 或者（如果应当通过 SQLParamData() 和 SQLPutData() 发送值的话）SQL_DATA_AT_EXEC 的相应长度值。 • SQL_PARAM_INPUT_OUTPUT: 参数标记与调用（CALL）的存储过程的输入／输出参数相关联。 执行语句时, 将把参数的实际数据值发送至服务器: *ParameterValuePtr* 缓冲区必须包含有效的输入数据值; *StrLen_or_IndPtr* 缓冲区必须包含 SQL_NTS、SQL_NULL_DATA 或者（如果应当通过 SQLParamData() 和 SQLPutData() 发送值的话）SQL_DATA_AT_EXEC 的相应长度值。 • SQL_PARAM_OUTPUT: 参数标记与调用（CALL）的存储过程的输出参数或该存储过程的返回值相关联。 在执行语句之后, 将把输出参数的数据返回到 *ParameterValuePtr* 和 *StrLen_or_IndPtr* 指定的应用程序缓冲区, 除非它们都是空指针, 如果是那样, 将把输出数据废弃。如果输出参数不带返回值, 则将 *StrLen_or_IndPtr* 设置为 SQL_NULL_DATA。

表 22. SQLBindParameter 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT	ValueType	输入	参数的 C 数据类型。支持下列类型: • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_BIGINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_BLOB_LOCATOR • SQL_CLOB_LOCATOR • SQL_DBCLOB_LOCATOR 指定 SQL_C_DEFAULT 将导致将数据由其缺省 C 数据类型转换为 *ParameterType* 指示的类型。
SQLSMALLINT	ParameterType	输入	参数的 SQL 数据类型。
SQLINTEGER	ColumnSize	输入	对应的参数标记的精度。如果 *ParameterType* 指示: • 二进制或单字节字符串（例如 SQL_CHAR），则此自变量是此参数标记的最大长度（字节）。 • 双字节字符串（例如 SQL_GRAPHIC），则此自变量是此参数的最大长度（以双字节字符数计）。 • SQL_DECIMAL 或 SQL_NUMERIC，则此自变量是最大小数精度。 • 否则，忽略此自变量。
SQLSMALLINT	DecimalDigits	输入	相应参数的标度（如果 *ParameterType* 是 SQL_DECIMAL 或 SQL_NUMERIC 的话）。如果 *ParameterType* 是 SQL_TYPE_TIMESTAMP，则这是时间戳记的字符表示法中小数点右边的位数（例如，yyyy-mm-dd hh:mm:ss.fff 的标度是 3）。 除了用于此处提到的 *ParameterType* 值之外，忽略 *DecimalDigits*。

SQLBindParameter

表 22. SQLBindParameter 自变量 (续)

数据类型	自变量	用途	描述
SQLPOINTER	ParameterValuePtr	输入（延迟）和/或输出（延迟）	• 对于输入（*InputOutputType* 设置为 SQL_PARAM_INPUT 或 SQL_PARAM_INPUT_OUTPUT）： 在执行时，如果 *StrLen_or_IndPtr* 不包含 SQL_NULL_DATA 或 SQL_DATA_AT_EXEC，则 *ParameterValuePtr* 指向包含参数的实际数据的缓冲区。 如果 *StrLen_or_IndPtr* 包含 SQL_DATA_AT_EXEC，则 *ParameterValuePtr* 是应用程序定义的与此参数相关联的 32 位值。这个 32 位值通过后续的 SQLParamData() 调用返回给应用程序。 如果调用 SQLParamOptions() 来对参数指定多个值，则 *ParameterValuePtr* 是指向 *BufferLength* 值的输入缓冲区数组的指针。 • 对于输出（*InputOutputType* 设置为 SQL_PARAM_OUTPUT 或 SQL_PARAM_INPUT_OUTPUT）： *ParameterValuePtr* 指向将用来存储存储过程的输出参数的缓冲区。 如果 *InputOutputType* 设置为 SQL_PARAM_OUTPUT，并且 *ParameterValuePtr* 和 *StrLen_or_IndPtr* 都是空指针，则废弃存储过程的输出参数值或返回值。
SQLINTEGER	BufferLength	输入	不使用。

表 22. SQLBindParameter 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER *	StrLen_or_IndPtr	输入（延迟）和 / 或输出（延迟）	如果这是输入或输入 / 输出参数: 这是指向一个位置的指针，（在执行语句时，）该位置包含存储在 *ParameterValuePtr* 处的参数标记值的长度。 要对参数标记指定空值，此存储器位置必须包含 SQL_NULL_DATA。 如果 *ValueType* 是 SQL_C_CHAR，则此存储位置必须包含存储在 *ParameterValuePtr* 处的数据的精确长度，或者，如果 *ParameterValuePtr* 处的内容是以空终止的，则必须包含 SQL_NTS。 如果 *ValueType* 指示 LOB 数据，则此存储位置必须包含存储在 *ParameterValuePtr* 处的数据的长度。 如果 *ValueType* 指示字符数据（显式指定，或使用 SQL_C_DEFAULT 隐式指定），并且此指针设置为空，则假定应用程序总是在 *ParameterValuePtr* 中提供以空终止的字符串。这还暗示此参数标记永远不会具有空值。 在调用 `SQLExecute()` 或 `SQLExecDirect()` 时，并且 *StrLen_or_IndPtr* 指向 SQL_DATA_AT_EXEC 值时，将使用 `SQLPutData()` 来发送数据。此参数称为**执行数据**参数。

用法

参数标记由 SQL 语句中的 “?” 字符表示，它用来指示语句中的一个位置，执行语句时，将在该位置替代应用程序提供的值。这个值是从应用程序变量获取的。

在执行 SQL 语句之前，应用程序必须将变量绑定到 SQL 语句中的每个参数标记。对于此函数，*ParameterValuePtr* 和 *StrLen_or_IndPtr* 是延迟自变量；在执行语句时，存储位置必须有效并且包含输入数据值。这意味着应该将 `SQLExecDirect()` 或 `SQLExecute()` 调用与 `SQLBindParameter()` 调用置于同一个过程作用域中，或者，必须动态地分配或静态地或全局地声明这些存储位置。

参数标记是通过编号（*ParameterNumber*）引用的，并且是从 1 开始从左到右按顺序编号的。

此函数绑定的所有参数将一直有效，这将持续到使用 SQL_DROP 或 SQL_RESET_PARAMS 选项调用 `SQLFreeStmt()` 为止，或持续到对同一个参数编号 *ParameterNumber* 再次调用 `SQLBindParameter()` 为止。

在执行 SQL 语句且处理结果之后，应用程序可能希望重新使用该语句句柄来执行另一个 SQL 语句。如果参数标记规格不同（参数数目、长度或类型），则应使用 SQL_RESET_PARAMS 来调用 `SQLFreeStmt()` 以重设或清除参数绑定。

ValueType 给出的 C 缓冲区数据类型必须与 *ParameterType* 指示的 SQL 数据类型相兼容，否则会出错。

SQLBindParameter

应用程序可以在 *ParameterValuePtr* 缓冲区中传送参数值，也可以通过调用一次或多次 SQLPutData() 来进行传送。在后一种情况下，这些参数是执行数据参数。应用程序通过将 SQL_DATA_AT_EXEC 值放入 *StrLen_or_IndPtr* 缓冲区来将执行数据参数的情况通知 DB2 UDB CLI。它将 *ParameterValuePtr* 输入自变量设置成一个 32 位值，后续的 SQLParamData() 调用将返回这个值，并且这个值可用来标识参数位置。

由于在执行语句之前不会对 *ParameterValuePtr* 和 *StrLen_or_IndPtr* 引用的变量中的数据进行验证，所以在调用 SQLExecute() 或 SQLExecDirect() 之前不会检测或报告数据内容或格式错误。

SQLBindParameter() 在实质上是通过提供用于指定参数是输入、输入和输出或输出的方法来扩展 SQLSetParam() 函数的功能。要正确地处理存储过程的参数，此信息是必需的。

InputOutputType 自变量指定参数的类型。不调用过程的 SQL 语句中的所有参数都是输入参数。存储过程调用中的参数可以是输入、输入／输出或输出参数。虽然 DB2 存储过程自变量约定通常暗示所有过程自变量都是输入／输出自变量，但应用程序员仍可以选择在 SQLBindParameter() 上更确切地指定输入或输出性质以遵循更严格的编码样式。并且，请注意，这些类型应该与使用 SQL CREATE PROCEDURE 语句注册存储过程时指定的参数类型一致。

- 如果应用程序不能确定过程调用中的参数的类型，则将 *InputOutputType* 设置为 SQL_PARAM_INPUT; 如果数据源对该参数返回一个值，则 DB2 UDB CLI 将其废弃。
- 如果应用程序将一个参数标记为 SQL_PARAM_INPUT_OUTPUT 或 SQL_PARAM_OUTPUT，并且数据源没有返回值，DB2 UDB CLI 就会将 *StrLen_or_IndPtr* 缓冲区设置为 SQL_NULL_DATA。
- 如果应用程序将参数标记为 SQL_PARAM_OUTPUT，则在处理 CALL 语句之后将参数的数据返回给应用程序。如果 *ParameterValuePtr* 和 *StrLen_or_IndPtr* 自变量都是空指针，则 DB2 UDB CLI 废弃输出值。如果数据源没有对某个输出参数返回值，DB2 UDB CLI 就会将 *StrLen_or_IndPtr* 缓冲区设置为 SQL_NULL_DATA。
- 对于此函数，*ParameterValuePtr* 和 *StrLen_or_IndPtr* 都是延迟自变量。对于 *InputOutputType* 设置为 SQL_PARAM_INPUT 或 SQL_PARAM_INPUT_OUTPUT 的情况，在执行语句时，存储位置必须有效并且包含输入数据值。这意味着应该将 SQLExecDirect() 或 SQLExecute() 调用与 SQLBindParameter() 调用置于同一个过程作用域中，或者，必须动态地分配或静态地／全局地声明这些存储位置。

  同样，如果 *InputOutputType* 设置为 SQL_PARAM_OUTPUT 或 SQL_PARAM_INPUT_OUTPUT，则在执行 CALL 语句之前，ParameterValuePtr 和 StrLen_or_IndPtr 缓冲区位置必须保持有效。

应用程序可以在 *ParameterValuePtr* 缓冲区中传送参数值，也可以通过调用一次或多次 SQLPutData() 来进行传送。在后一种情况下，这些参数是执行数据参数。应用程序通过将 SQL_DATA_AT_EXEC 值放入 StrLen_or_IndPtr 缓冲区来将执行数据参数的情况通知 DB2 UDB CLI。它将 *ParameterValuePtr* 输入自变量设置成一个 32 位值，后续的 SQLParamData() 调用将返回这个值，并且这个值可用来标识参数位置。

当使用 SQLBindParameter() 来将应用程序变量绑定到存储过程的输出参数时，如果在内存中将 *ParameterValuePtr* 缓冲区相邻地放在 *StrLen_or_IndPtr* 缓冲区之后，则 DB2 UDB CLI 能提供一定程度的性能增强。例如:

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

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 23. SQLBindParameter SQLSTATE

SQLSTATE	描述	说明
07006	转换无效	从 *ValueType* 自变量标识的数据值到 *ParameterType* 自变量标识的数据值的转换不是有意义的转换。（例如，从 SQL_C_DATE 到 SQL_DOUBLE 的转换是没有意义的。）
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
58004	意外的系统故障	不可恢复的系统错误。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY003	程序类型超出范围	*ParameterNumber* 自变量指定的值不具有有效的数据类型，也不是 SQL_C_DEFAULT。
HY004	SQL 数据类型超出范围	对 *ParameterType* 自变量指定的值不是有效的 SQL 数据类型。
HY009	自变量值无效	*ParameterValuePtr* 自变量是空指针，*StrLen_or_IndPtr* 自变量是空指针，但 *InputOutputType* 不是 SQL_PARAM_OUTPUT。
HY010	函数顺序错误	在 `SQLExecute()` 或 `SQLExecDirect()` 返回 SQL_NEED_DATA 后调用了此函数，但还没有为所有*执行数据*参数发送数据。
HY013	意外的内存处理错误	DB2 UDB CLI 无法访问支持此函数的执行或完成所必需的内存。
HY014	句柄太多	已分配了最大数目的句柄
HY021	描述符信息不一致	一致性检查期间检查的描述符信息不一致。
HY090	字符串或缓冲区长度无效	对自变量 *BufferLength* 指定的值小于 0。
HY093	参数号无效	对自变量 *ValueType* 指定的值小于 1 或大于服务器所支持的最大参数数目。
HY094	标度值无效	对 *ParameterType* 指定的值是 SQL_DECIMAL 或 SQL_NUMERIC，而对 *DecimalDigits* 指定的值小于 0 或大于对自变量 *ParamDef*（精度）指定的值。 对 *ParameterType* 指定的值是 SQL_C_TIMESTAMP，*ParameterType* 的值是 SQL_CHAR 或 SQL_VARCHAR，而 *DecimalDigits* 的值小于 0 或大于 6。
HY104	精度值无效	对 *ParameterType* 指定的值是 SQL_DECIMAL 或 SQL_NUMERIC，而对 *ParamDef* 指定的值小于 1。
HY105	参数类型无效	*InputOutputType* 是 SQL_PARAM_INPUT、SQL_PARAM_OUTPUT 或 SQL_PARAM_INPUT_OUTPUT 的其中之一。
HYC00	驱动程序不具有能力	DB2 UDB CLI 或数据源不支持由对自变量 *ValueType* 指定的值和对自变量 *ParameterType* 指定的值的组合所指定的转换。 DB2 UDB CLI 或数据源不支持对 *ParameterType* 自变量指定的值。

参考

- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 182 页的『SQLParamData — 获取下一个需要其数据值的参数』
- 第 201 页的『SQLPutData — 传送参数的数据值』

SQLCancel — 取消语句

用途

SQLCancel() 尝试结束正在以异步方式执行的 SQL 语句操作的处理。

SQLCancel() 仅用于兼容性目的，对 SQL 语句执行不起任何作用。

语法

```
SQLRETURN SQLCancel (SQLHSTMT     hstmt);
```

函数自变量

表 24. SQLCancel 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄

用法

成功的返回码指示实现已接受取消请求；并不确保能够取消处理。

返回码

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

诊断

表 25. SQLCancel SQLSTATE

SQLSTATE	描述	说明
HY009 *****	自变量值无效	*hstmt* 不是语句句柄

限制

DB2 UDB CLI 不支持异步语句执行。

SQLCloseCursor — 关闭游标语句

用途

SQLCloseCursor() 关闭在语句句柄上打开的游标。

语法

```
SQLRETURN SQLCloseCursor (SQLHSTMT     hstmt);
```

函数自变量

表 26. SQLCancel 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*hstmt*	输入	语句句柄

用法

调用 SQLCloseCursor() 将关闭任何与语句句柄相关联的游标并废弃任何暂挂结果。如果没有打开的游标与语句句柄相关联，则此函数没有任何效果。

如果语句句柄引用具有多个结果集的存储过程，则 SQLCloseCursor() 只关闭当前结果集。任何附加的结果集都保持打开并且可使用。

返回码

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

诊断

表 27. SQLCancel SQLSTATE

SQLSTATE	描述	说明
08003 *	连接未打开	未建立 hstmt 的连接
HY009 *	自变量值无效	*hstmt* 不是语句句柄

SQLColAttributes — 列属性

用途

SQLColAttributes() 获取结果集的列的属性，并且还用来确定列数。SQLColAttributes() 是 SQLDescribeCol() 函数的更具扩展性的备用函数。

在调用此函数之前，必须调用 SQLPrepare() 或 SQLExecDirect()。

如果应用程序不了解列的各种属性（如数据类型和长度），则必须在 SQLBindCol() 之前调用此函数（或 SQLDescribeCol()）。

语法

```
SQLRETURN SQLColAttributes (SQLHSTMT       hstmt,
                            SQLSMALLINT    icol,
                            SQLSMALLINT    fDescType,
                            SQLCHAR        *rgbDesc,
                            SQLINTEGER     cbDescMax,
                            SQLINTEGER     *pcbDesc,
                            SQLINTEGER     *pfDesc);
```

函数自变量

表 28. SQLColAttributes 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*icol*	输入	结果集中的列号（必须介于 1 与结果集中的列数之间，包括 1 与列数）。当指定了 SQL_DESC_COUNT 时，将忽略此自变量。
SQLSMALLINT	*fDescType*	输入	表 29 描述了支持的值。
SQLCHAR *	*rgbDesc*	输出	指向字符串列属性的缓冲区的指针
SQLINTEGER	*cbDescMax*	输入	描述符缓冲区（*rgbDesc*）的长度
SQLINTEGER *	*pcbDesc*	输出	要返回的描述符中的实际字节数。如果此自变量包含等于或大于 *rgbDesc* 缓冲区的长度的值，则将发生截断。将把描述符截断成长度为 *cbDescMax* - 1 字节。
SQLINTEGER *	*pfDesc*	输出	指向用于存放关于数字列属性的信息的整数的指针。

表 29. fDescType 描述符类型

描述符	类型	描述
SQL_DESC_COUNT	SMALLINT	在 *pfDesc* 中返回结果集中的列数。
SQL_DESC_NAME	CHAR(128)	在 *rgbDesc* 中返回列 *icol* 的名称。如果该列是表达式，则返回的结果随特定产品的不同而不同。
SQL_DESC_TYPE	SMALLINT	在 *pfDesc* 中返回 *icol* 中标识的列的 SQL 数据类型。第 18 页的表 5 列示了 *pfSqlType* 的可能值。

表 29. fDescType 描述符类型 (续)

描述符	类型	描述
SQL_DESC_LENGTH	INTEGER	在 *pfDesc* 中返回与列相关联的数据的字节数。 如果 *icol* 中标识的列基于字符，例如 SQL_CHAR、SQL_VARCHAR 或 SQL_LONG_VARCHAR，则返回实际长度或最大长度。 如果列类型是 SQL_DECIMAL 或 SQL_NUMERIC，则 SQL_DESC_LENGTH 将是 *(精度 * 256) + 标度*。返回此项的目的是可以将这个值作为输入来在 `SQLBindCol()` 上传送。对于这些数据类型，也可以使用 SQL_DESC_PRECISION 和 SQL_DESC_SCALE 来将精度和标度作为单独的值获取。
SQL_DESC_PRECISION	SMALLINT	返回列的精度属性。
SQL_DESC_SCALE	SMALLINT	返回列的标度属性。
SQL_DESC_NULLABLE	SMALLINT	如果 *icol* 标识的列可以包含空值，则在 *pfDesc* 中返回 SQL_NULLABLE。 如果将列约束为不接受空值，则在 *pfDesc* 中返回 SQL_NO_NULLS。
SQL_DESC_UNNAMED	SMALLINT	如果 NAME 字段是实际的名称，则此项为 SQL_NAMED，如果 NAME 字段是由实现生成的名称，则此项为 SQL_UNNAMED。
SQL_DESC_AUTO_INCREMENT	INTEGER	如果在将新行插入表之后列可以自动递增，则此项为 SQL_TRUE。如果列不能自动递增，则为 SQL_FALSE。
SQL_DESC_SEARCHABLE	INTEGER	如果列不能在 **WHERE** 子句中使用，则为 SQL_UNSEARCHABLE。 如果列只能与 **LIKE** 谓词一起在 **WHERE** 子句中使用，则为 SQL_LIKE_ONLY。 如果列可以与除 **LIKE** 之外的所有比较运算符一起在 **WHERE** 子句中使用，则为 SQL_ALL_EXCEPT_LIKE。 如果列可以与任何比较运算符一起在 **WHERE** 子句中使用，则为 SQL_SEARCHABLE 要检索此属性，必须已对语句句柄或连接句柄将属性 SQL_ATTR_EXTENDED_COL_INFO 设置为 SQL_TRUE。

表 29. fDescType 描述符类型 (续)

描述符	类型	描述
SQL_DESC_UPDATABLE	INTEGER	已定义的常量的值对列进行描述: SQL_ATTR_READONLY SQL_ATTR_WRITE SQL_ATTR_READWRITE_UNKNOWN SQL_COLUMN_UPDATABLE 描述结果集中的列的可更新能力。列是否可更新可以基于数据类型、用户特权和结果集本身的定义。如果不清楚列是否可更新，则应返回 SQL_ATTR_READWRITE_UNKNOWN。 要检索此属性，必须已对语句句柄或连接句柄将属性 SQL_ATTR_EXTENDED_COL_INFO 设置为 SQL_TRUE。
SQL_DESC_BASE_TABLE	CHAR(128)	构建此列所基于的底层表的名称。 要检索此属性，必须已对语句句柄或连接句柄将属性 SQL_ATTR_EXTENDED_COL_INFO 设置为 SQL_TRUE。
SQL_DESC_BASE_COLUMN	CHAR(128)	构建此列所基于的底层表中的实际列的名称。 要检索此属性，必须已对语句句柄或连接句柄将属性 SQL_ATTR_EXTENDED_COL_INFO 设置为 SQL_TRUE。
SQL_DESC_BASE_SCHEMA	CHAR(128)	构建此列所基于的底层表的模式名。 要检索此属性，必须已对语句句柄或连接句柄将属性 SQL_ATTR_EXTENDED_COL_INFO 设置为 SQL_TRUE。
SQL_DESC_LABEL	CHAR(128)	此列的标号，如果存在的话。否则，是长度为零的字符串。 要检索此属性，必须已对语句句柄或连接句柄将属性 SQL_ATTR_EXTENDED_COL_INFO 设置为 SQL_TRUE。

用法

与返回特定自变量集的 SQLDescribeCol() 不同，SQLColAttributes() 可用来指定要对特定列接收哪个属性。如果期望的信息是字符串，则在 *rgbDesc* 中返回它。如果期望的信息是数字，则在 *pfDesc* 中返回它。

虽然 SQLColAttributes() 允许将来进行扩展，但与 SQLDescribeCol() 相比，它要求为每个列进行更多的调用才能接收到相同的信息。

如果 *fDescType* 描述符类型不适用于数据库服务器，则在 *rgbDesc* 中返回空字符串，或在 *pfDesc* 中返回零，这取决于描述符的期望结果。

列由编号标识（从 1 开始从左到右按顺序编号），并可以按任何次序描述。

在 *fDescType* 设置为 SQL_DESC_COUNT 的情况下调用 SQLColAttributes() 是调用 SQLNumResultCols() 来确定是否可返回任何列的备用方法。

在调用 SQLColAttributes() 之前，调用 SQLNumResultCols() 来确定结果集是否存在。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

诊断

表 30. SQLColAttributes SQLSTATE

SQLSTATE	描述	说明
07009	列号无效	对自变量 *icol* 指定的值小于 1。
HY009	自变量值无效	对自变量 *fDescType* 指定的值不等于第 58 页的表 29 中指定的值。 自变量 *rgbDesc*、*pcbDesc* 或 *pfDesc* 是空指针。
HY010	函数顺序错误	在对 *hstmt* 调用 SQLPrepare() 或 SQLExecDirect() 之前调用了此函数。
HYC00	驱动程序不具有能力	DB2 UDB CLI 不识别数据库服务器对列 *icol* 返回的 SQL 数据类型。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 186 页的『SQLPrepare — 准备语句』

SQLColumnPrivileges — 获取与表列相关联的特权

用途

SQLColumnPrivileges() 返回所指定的表的列及相关联特权的列表。将在 SQL 结果集中返回此信息，可以使用那些用来处理从查询生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLColumnPrivileges  (
                SQLHSTMT            StatementHandle,
                SQLCHAR             *CatalogName,
                SQLSMALLINT         NameLength1,
                SQLCHAR             *SchemaName,
                SQLSMALLINT         NameLength2,
                SQLCHAR             *TableName
                SQLSMALLINT         NameLength3,
                SQLCHAR             *ColumnName,
                SQLSMALLINT         NameLength4);
```

函数自变量

表 31. SQLColumnPrivileges 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*语句句柄*	输入	语句句柄
SQLCHAR *	*CatalogName*	输入	由三部分组成的表名的目录限定符。这必须是空指针或长度为零的字符串。
SQLSMALLINT	*NameLength1*	输入	*CatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	*SchemaName*	输入	表名的模式限定符。
SQLSMALLINT	*NameLength2*	输入	*SchemaName* 的长度
SQLCHAR *	*TableName*	输入	表名。
SQLSMALLINT	*NameLength3*	输入	*TableName* 的长度。
SQLCHAR *	*ColumnName*	输入	缓冲区，它可以包含*模式值*，以通过列名限定结果集。
SQLSMALLINT	*NameLength4*	输入	*ColumnName* 的长度。

用法

结果是作为包含第 63 页的表 32 中列示的列的标准结果集返回的。此结果集按 TABLE_CAT、TABLE_SCHEM、TABLE_NAME、COLUMN_NAME 和 PRIVILEGE 进行排序。如果有多个特权与任何给定的列相关联，则将每个特权作为单独的行返回。典型的应用程序可能希望在调用 SQLColumns() 之后调用此函数来确定列特权信息。应用程序应使用 SQLColumns() 结果集的 TABLE_SCHEM、TABLE_NAME 和 COLUMN_NAME 列中返回的字符串来作为此函数的输入自变量。

由于在许多情况下调用 SQLColumnPrivileges() 都意味着要对系统目录进行复杂从而成本高昂的查询，所以，您应该有节制地使用这些调用，并且应该保存结果而不是重复地进行调用。

为了与 SQL92 限制一致，编目函数结果集的 VARCHAR 列是使用最大长度属性 128 声明的。由于 DB2 名称的长度小于 128，所以应用程序可以选择始终为输出缓冲区留出 128 个字符（加上空终止符）的空间，也可以选择使用

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_LEN、SQL_MAX_TABLE_NAME_LEN 和 SQL_MAX_COLUMN_NAME_LEN 来调用 SQLGetInfo() 以分别确定所连接的 DBMS 支持的 TABLE_CAT、TABLE_SCHEM、TABLE_NAME 和 COLUMN_NAME 列的实际长度。

注意，*ColumnName* 自变量接受搜索模式。

虽然在将来的发行版中可能会添加新列和更改现有列的名称，但当前列的位置不会更改。

表 32. SQLColumnPrivileges 返回的列

列号 / 名称	数据类型	描述
TABLE_CAT	VARCHAR(128)	此项始终为空。
TABLE_SCHEM	VARCHAR(128)	包含 TABLE_NAME 的模式的名称。
TABLE_NAME	VARCHAR(128) 非空	表或视图的名称。
COLUMN_NAME	VARCHAR(128) 非空	指定的表或视图的列的名称。
GRANTOR	VARCHAR(128)	进行特权授权的用户的授权标识。
GRANTEE	VARCHAR(128)	接受特权授权的用户的授权标识。
PRIVILEGE	VARCHAR(128)	列特权。这可以是: • INSERT • REFERENCES • SELECT • UPDATE
IS_GRANTABLE	VARCHAR(3)	指示是否允许被授权人将特权授予其它用户。 值为 YES 或 NO。

注意: DB2 CLI 使用的列名遵循 X/Open CLI CAE 规范样式。列类型、内容和次序与 ODBC 中对 SQLColumnPrivileges() 结果集定义的那些列类型、内容和次序完全相同。

如果有多个特权与列相关联，则将每个特权都作为结果集中的单独行返回。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 33. SQLColumnPrivileges SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。

表 33. SQLColumnPrivileges SQLSTATE (续)

SQLSTATE	描述	说明
HY010	函数顺序错误	语句句柄的游标已打开。 没有用于此语句句柄的连接。

限制

无

示例

```
/* From the CLI sample TBINFO.C */
/* ... */

    /* call SQLColumnPrivileges */
    printf("\n Call SQLColumnPrivileges for:\n");
    printf(" tbSchema = %s\n", tbSchema);
    printf(" tbName = %s\n", tbName);
    sqlrc = SQLColumnPrivileges( hstmt, NULL, 0,
                                 tbSchema, SQL_NTS,
                                 tbName, SQL_NTS,
                                 colNamePattern, SQL_NTS);
```

参考

- 第 65 页的『SQLColumns — 获取表的列信息』
- 第 237 页的『SQLTables — 获取表信息』

SQLColumns — 获取表的列信息

用途

SQLColumns() 返回所指定的表中的列的列表。将在 SQL 结果集中返回此信息，可以使用那些用来取装由 SELECT 语句生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLColumns  (SQLHSTMT       hstmt,
                       SQLCHAR        *szCatalogName,
                       SQLSMALLINT    cbCatalogName,
                       SQLCHAR        *szSchemaName,
                       SQLSMALLINT    cbSchemaName,
                       SQLCHAR        *szTableName,
                       SQLSMALLINT    cbTableName,
                       SQLCHAR        *szColumnName,
                       SQLSMALLINT    cbColumnName);
```

函数自变量

表 34. SQLColumns 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLCHAR *	*szCatalogName*	输入	缓冲区，它可能包含*模式值*，以限定结果集。*目录*是由三部分组成的表名的第一部分。 这必须是空指针或长度为零的字符串。
SQLSMALLINT	*cbCatalogName*	输入	*szCatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	*szSchemaName*	输入	缓冲区，它可能包含*模式值*，以通过模式名限定结果集。
SQLSMALLINT	*cbSchemaName*	输入	*szSchemaName* 的长度
SQLCHAR *	*szTableName*	输入	缓冲区，它可能包含*模式值*，以通过表名限定结果集。
SQLSMALLINT	*cbTableName*	输入	*szTableName* 的长度
SQLCHAR *	*szColumnName*	输入	缓冲区，它可能包含*模式值*，以通过列名限定结果集。
SQLSMALLINT	*cbColumnName*	输入	*szColumnName* 的长度

用法

此函数检索关于表或表列表的列的信息。

SQLColumns() 返回标准结果集。第 66 页的表 35 列示了结果集中的列。应用程序应预期在将来的发行版中可能会在 REMARKS 列之后添加附加的列。

szCatalogName、*szSchemaName*、*szTableName* 和 *szColumnName* 自变量接受搜索模式。可以随通配符一起指定转义字符，以便可以在搜索模式中使用实际字符。转义字符是在 SQL_ATTR_ESCAPE_CHAR 环境属性中指定的。

SQLColumns

此函数不返回有关结果集中的列的信息，可使用 SQLDescribeCol() 或 SQLColAttributes() 检索该信息。如果应用程序想要获取结果集的列信息，则它始终应该调用 SQLDescribeCol() 或 SQLColAttributes() 以提高效率。SQLColumns() 意味着对系统目录执行复杂的查询，这可能需要大量的系统资源。

表 35. SQLColumns 返回的列

列名	数据类型	描述
TABLE_CAT	VARCHAR(128)	当前服务器。
TABLE_SCHEM	VARCHAR(128)	包含 TABLE_NAME 的模式的名称。
TABLE_NAME	VARCHAR(128)	表或视图的名称
COLUMN_NAME	VARCHAR(128)	列标识符。所指定的表或视图的列的名称。
DATA_TYPE	SMALLINT 非空	标识列的 SQL 数据类型。
TYPE_NAME	VARCHAR(128) 非空	表示与 DATA_TYPE 相对应的数据类型的名称的字符串。
LENGTH_PRECISION	INTEGER	如果 DATA TYPE 是近似数字数据类型，则此列包含列的尾数精度的位数。对于精确数字数据类型，此列包含列中允许的小数位的总位数。对于时间和时间戳记数据类型，此列包含小数秒组件的精度的位数；否则，此列为空。 **注意**：精度的 ODBC 定义通常是用来存储数据类型的位数。
BUFFER_LENGTH	INTEGER	如果在 SQLBindCol()、SQLGetData() 和 SQLBindParam() 调用中指定了 SQL_DEFAULT，则此项是用于存储此列中的数据的最大字节数。
NUM_SCALE	SMALLINT	列的标度。对于标度不适用的数据类型，将返回空。
NUM_PREC_RADIX	SMALLINT	10、2 或空。如果 DATA_TYPE 是近似数字数据类型，则此列包含值 2，这样，LENGTH_PRECISION 列包含列中允许的位数。 如果 DATA_TYPE 是精确数字数据类型，则此列包含值 10，并且 LENGTH_PRECISION 和 NUM_SCALE 列包含列所允许的小数位数。 对于数字数据类型，DBMS 可以返回值为 10 或 2 的 NUM_PREC_RADIX。 对于基数不适用的数据类型，将返回空。
NULLABLE	SMALLINT 非空	如果列不接受空值，则此项为 SQL_NO_NULLS。 如果列接受空值，则为 SQL_NULLABLE。
REMARKS	VARCHAR(254)	可能包含关于列的描述性信息。

表 35. SQLColumns 返回的列 (续)

列名	数据类型	描述
COLUMN_DEF	VARCHAR(254)	列的缺省值。如果缺省值是数字文字，则此列包含数字文字的字符表示法，并且不用单引号括起来。如果缺省值是字符串，则此列是用单引号括起来的该字符串。如果缺省值是*伪*文字，如对于 DATE、TIME 和 TIMESTAMP 列，则此列包含伪文字的关键字（例如 CURRENT DATE），并且不用引号括起来。 如果指定了 NULL 作为缺省值，则此列将返回空字，并且不用引号括起来。如果不进行截断就不能表示缺省值，则此列包含 TRUNCATED，并且不用单引号括起来。如果没有指定缺省值，则此列为空。
DATETIME_CODE	INTEGER	此列当前为空。
CHAR_OCTET_LENGTH	INTEGER	包含字符数据类型列的最大长度（八位元）。对于“单字节”字符集，这与 LENGTH_PRECISION 相同。对于所有其它数据类型，这是空。
ORDINAL_POSITION	INTEGER 非空	列在表中的序数位置。表中的第一列编号为 1。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 36. SQLColumns SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HY010	函数顺序错误	语句句柄的游标已打开。 没有用于此语句句柄的连接。

SQLConnect — 连接到数据源

用途

SQLConnect() 建立与目标数据库的连接。应用程序必须提供目标 SQL 数据库，并可选择提供授权名和授权字符串。

在调用此函数之前，必须调用 SQLAllocConnect()。

在调用 SQLAllocStmt() 之前，必须调用此函数。

语法

```
SQLRETURN SQLConnect (SQLHDBC         hdbc,
                      SQLCHAR         *szDSN,
                      SQLSMALLINT     cbDSN,
                      SQLCHAR         *szUID,
                      SQLSMALLINT     cbUID,
                      SQLCHAR         *szAuthStr,
                      SQLSMALLINT     cbAuthStr);
```

函数自变量

表 37. SQLConnect 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄
SQLCHAR *	*szDSN*	输入	数据源: 数据库的名称或别名名称
SQLSMALLINT	*cbDSN*	输入	*szDSN* 自变量的内容的长度
SQLCHAR *	*szUID*	输入	授权名（用户标识符）
SQLSMALLINT	*cbUID*	输入	*szUID* 自变量的内容的长度
SQLCHAR *	*szAuthStr*	输入	授权字符串（密码）
SQLSMALLINT	*cbAuthStr*	输入	*szAuthStr* 自变量的内容的长度

用法

可以在应用程序中使用 SQLSetConnectOption() 来定义各种连接特征（选项）。

可以将 SQLConnect() 的输入长度自变量（*cbDSN*、*cbUID* 和 *cbAuthStr*）设置为与它们相关联的数据的实际长度。这不包括任何空终止字符或 SQL_NTS（用于指示相关联的数据以空终止）。

除非将 *szDSN* 和 *szUID* 自变量值括在引号中，否则在处理它们之前将除去它们的前导和尾部空格。

必须已在系统上定义了数据源才能使连接起作用。在 iSeries 上，可以使用“使用关系数据库目录项”（WRKRDBDIRE）命令来确定已定义了哪些数据源以及定义附加的数据源（可选）。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 38. SQLConnect SQLSTATE

SQLSTATE	描述	说明
08001	无法连接到数据源	驱动程序无法建立与数据源（服务器）的连接。
08002	连接在使用中	已使用指定的 *hdbc* 来建立与数据源的连接，并且该连接仍是打开的。
08004	数据源已拒绝建立连接	数据源（服务器）已拒绝建立连接。
28000	授权说明无效	对自变量 *szUID* 指定的值或对自变量 *szAuthStr* 指定的值违反了数据源定义的限制。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	对自变量 *cbUID* 指定的值小于 0，但不等于 SQL_NTS，而自变量 *szUID* 不是空指针。 对自变量 *cbUID* 指定的值小于 0，但不等于 SQL_NTS，而自变量 *szUID* 不是空指针。 对自变量 *cbAuthStr* 指定的值小于 0，但不等于 SQL_NTS，而自变量 *szAuthStr* 不是空指针。 在 *szDSN*、*szUID* 或 *szAuthStr* 自变量中找到不匹配的双引号（"）。
HY013 *****	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。
HY501 *****	数据源名无效	在自变量 *szDSN* 中指定了无效的数据源名。

限制

不支持 IBM DBMS 的隐式连接（或缺省数据库）选项。在可以执行任何 SQL 语句之前，必须调用 SQLConnect()。iSeries 不支持在一个作业中同时与同一个数据源建立多个连接。

当您在较新版本上使用 DB2 UDB CLI 时，SQLConnect() 可能会遇到 SQL0144 消息。此消息指示数据源（服务器）带有必须删除的旧 SQL 软件包。要删除这些软件包，请在服务器系统上运行以下命令:

```
DLTSQLPKG SQLPKG(QGPL/QSQCLI*)
```

下一个 SQLConnect() 将创建新的 SQL 软件包。

示例

请参考 SQLAllocEnv()（第 28 页的『示例』）。

参考

- 第 24 页的『SQLAllocConnect — 分配连接句柄』
- 第 32 页的『SQLAllocStmt — 分配语句句柄』

SQLCopyDesc — 复制描述语句

用途

SQLCopyDesc() 将与源句柄相关联的数据结构的字段复制到与目标句柄相关联的数据结构中。

除了不更改 ALLOC_TYPE 字段之外，将覆盖与目标句柄相关联的数据结构中的任何现有数据。

语法

```
SQLRETURN SQLCopyDesc (SQLHDESC     sDesc)
                      (SQLHDESC     tDesc);
```

函数自变量

表 39. SQLCancel 自变量

数据类型	自变量	用途	描述
SQLHDESC	*sDesc*	输入	源描述符句柄
SQLHDESC	*tDesc*	输入	目标描述符句柄

用法

可以通过调用 GetStmtAttr() 来获取语句的自动生成的行和参数描述符的句柄。

返回码

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

SQLDataSources — 获取数据源列表

用途

SQLDataSources() 返回可用的目标数据库的列表，并且每次返回一个目标数据库。为了使数据库可用，必须对其进行编目。有关编目的更多信息，请参考 SQLConnect() 的使用说明或查看 “使用关系数据库（RDB）目录项”（WRKRDBDIRE）命令的联机帮助。

在建立连接之前，通常调用 SQLDataSources() 来确定可以连接的数据库。

语法

```
SQLRETURN   SQLDataSources   (SQLHENV          EnvironmentHandle,
                              SQLSMALLINT      Direction,
                              SQLCHAR          *ServerName,
                              SQLSMALLINT      BufferLength1,
                              SQLSMALLINT      *NameLength1Ptr,
                              SQLCHAR          *Description,
                              SQLSMALLINT      BufferLength2,
                              SQLSMALLINT      *NameLength2Ptr);
```

函数自变量

表 40. SQLDataSources 自变量

数据类型	自变量	用途	描述
SQLHENV	*EnvironmentHandle*	输入	环境句柄。
SQLSMALLINT	*Direction*	输入	应用程序使用此自变量来请求列表中的第一个数据源名或列表中的下一个数据源名。*Direction* 只可以具有下列值: • SQL_FETCH_FIRST • SQL_FETCH_NEXT
SQLCHAR *	*ServerName*	输出	指向用于存放检索到的数据源名的缓冲区的指针。
SQLSMALLINT	*BufferLength1*	输入	*ServerName* 指向的缓冲区的最大长度。此自变量应小于或等于 SQL_MAX_DSN_LENGTH + 1。
SQLSMALLINT *	*NameLength1Ptr*	输出	指向一个位置的指针，这个位置将存储可以在 *ServerName* 中返回的最大字节数。
SQLCHAR *	*Description*	输出	指向一个缓冲区的指针，将在这个缓冲区中返回数据源的描述。DB2 UDB CLI 将返回与对 DBMS 编目的数据库相关联的 ***Comment*** 字段。
SQLSMALLINT	*BufferLength2*	输入	*Description* 缓冲区的最大长度。
SQLSMALLINT *	*NameLength2Ptr*	输出	指向一个位置的指针，此函数将在该位置中返回可以对数据源描述返回的实际字节数。

用法

应用程序可以随时调用此函数，并可以将 *Direction* 设置为 SQL_FETCH_FIRST 或 SQL_FETCH_NEXT。

如果指定 SQL_FETCH_FIRST，则将始终返回列表中的第一个数据库。

如果指定 SQL_FETCH_NEXT，则:

- 紧跟着 SQL_FETCH_FIRST 调用，将返回列表中的第二个数据库。
- 在任何其它 SQLDataSources() 调用之前，将返回列表中的第一个数据库。
- 当列表中没有更多的数据库时，返回 SQL_NO_DATA_FOUND。如果再次调用此函数，则返回第一个数据库。
- 在任何其它时候，将返回列表中的下一个数据库。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

错误状态

表 41. SQLDataSources SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	在自变量 *ServerName* 中返回的数据源名比自变量 *BufferLength1* 中指定的值长。自变量 *NameLength1Ptr* 包含完整数据源名的长度。（函数将返回 SQL_SUCCESS_WITH_INFO。） 在自变量 *Description* 中返回的数据源名比自变量 *BufferLength2* 中指定的值长。自变量 *NameLength2Ptr* 包含完整数据源描述的长度。（函数将返回 SQL_SUCCESS_WITH_INFO。）
58004	意外的系统故障	不可恢复的系统错误。
HY000	一般错误	发生错误，该错误没有特定的 SQLSTATE，并且没有为其定义特定的 SQLSTATE。SQLError() 在自变量 *ErrorMsg* 中返回的错误消息描述了此错误及其原因。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *ServerName*、*NameLength1Ptr*、*Description* 或 *NameLength2Ptr* 是空指针。 方向值无效。
HY013	意外的内存处理错误	DB2 UDB CLI 无法访问支持此函数的执行或完成所必需的内存。
HY103	方向选项超出范围	对自变量 *Direction* 指定的值不等于 SQL_FETCH_FIRST 或 SQL_FETCH_NEXT。

授权

无。

示例

```
/* From CLI sample datasour.c */
/* ... */

#include <stdio.h>
#include <stdlib.h>
#include <sqlcli1.h>
#include "samputil.h"          /* Header file for CLI sample code */
```

```
/* ... */

/************************************************************************
** main
** - initialize
** - terminate
************************************************************************/
int main() {

    SQLHANDLE henv ;
    SQLRETURN rc ;
    SQLCHAR source[SQL_MAX_DSN_LENGTH + 1], description[255] ;
    SQLSMALLINT buffl, desl ;

/* ... */

    /* allocate an environment handle */
    rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;

    /* list the available data sources (servers) */
    printf( "The following data sources are available:\n" ) ;
    printf( "ALIAS NAME                      Comment(Description)\n" ) ;
    printf( "-------------------------------------------------\n" ) ;

    while ( ( rc = SQLDataSources( henv,
                                   SQL_FETCH_NEXT,
                                   source,
                                   SQL_MAX_DSN_LENGTH + 1,
                                   &buffl,
                                   description,
                                   255,
                                   &desl
                                 )
            ) != SQL_NO_DATA_FOUND
          ) printf( "%-30s  %s\n", source, description ) ;

    rc = SQLFreeHandle( SQL_HANDLE_ENV,  henv ) ;
    if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;

    return( SQL_SUCCESS ) ;


}
```

参考

无。

SQLDescribeCol — 描述列属性

用途

SQLDescribeCol() 返回 SELECT 语句生成的结果集中的所指示列的结果描述符信息（列名、类型和精度）。

如果应用程序只需要描述符信息的一个属性，则可以使用 SQLColAttributes() 函数来代替 SQLDescribeCol()。有关更多信息，请参考第 58 页的『SQLColAttributes — 列属性』。

在调用此函数之前，必须调用 SQLPrepare() 或 SQLExecDirect()。

通常在 SQLBindCol() 之前调用此函数（或 SQLColAttributes()）。

语法

```
SQLRETURN SQLDescribeCol (SQLHSTMT       hstmt,
                          SQLSMALLINT    icol,
                          SQLCHAR        *szColName,
                          SQLSMALLINT    cbColNameMax,
                          SQLSMALLINT    *pcbColName,
                          SQLSMALLINT    *pfSqlType,
                          SQLINTEGER     *pcbColDef,
                          SQLSMALLINT    *pibScale,
                          SQLSMALLINT    *pfNullable);
```

函数自变量

表 42. SQLDescribeCol 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*icol*	输入	要描述的列号
SQLCHAR *	*szColName*	输出	指向列名缓冲区的指针
SQLSMALLINT	*cbColNameMax*	输入	*szColName* 缓冲区的大小
SQLSMALLINT *	*pcbColName*	输出	可以为 *szColName* 自变量返回的字节数。如果 *pcbColName* 大于或等于 *cbColNameMax*，则将列名（*szColName*）截断成长度为 *cbColNameMax - 1* 字节。
SQLSMALLINT *	*pfSqlType*	输出	列的 SQL 数据类型
SQLINTEGER *	*pcbColDef*	输出	数据库中定义的列精度。 如果 *fSqlType* 指示图形 SQL 数据类型，则此变量指示该列可以存放的最大双字节字符数。
SQLSMALLINT *	*pibScale*	输出	数据库中定义的列标度（只适用于 SQL_DECIMAL、SQL_NUMERIC 和 SQL_TIMESTAMP）。
SQLSMALLINT *	*pfNullable*	输出	指示此列是否允许空 • SQL_NO_NULLS • SQL_NULLABLE

用法

列由编号标识并从 1 开始从左到右按次序编号，并可以按任何顺序描述。

必须提供有效的指针和缓冲区空间来供 *szColName* 自变量使用。如果对其余任何指针自变量指定空指针，则 DB2 UDB CLI 假定应用程序不需要该信息，并且不返回任何内容。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

如果 `SQLDescribeCol()` 返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO，则调用 `SQLError()` 函数可获取下列其中一个 SQLSTATE。

表 43. SQLDescribeCol SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	在自变量 *szColName* 中返回的列名比自变量 *cbColNameMax* 中指定的值长。自变量 *pcbColName* 包含完整列名的长度。（函数将返回 SQL_SUCCESS_WITH_INFO。）
07005 *	不是 SELECT 语句	与 *hstmt* 相关联的语句未返回结果集。没有可描述的列。（首先调用 `SQLNumResultCols()` 以确定结果集中是否有任何行。）
07009	列号无效	对自变量 *icol* 指定的值小于 1。 对自变量 *icol* 指定的值大于结果集中的列数。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *cbColNameMax* 中指定的长度小于 1。 自变量 *szColName* 或 *pcbColName* 是空指针。
HY010	函数顺序错误	在对 *hstmt* 调用 `SQLPrepare()` 或 `SQLExecDirect()` 之前调用了此函数。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。
HYC00	驱动程序不具有能力	DB2 UDB CLI 不识别列 *icol* 的 SQL 数据类型。

示例

有关以下示例的完整列表，请参考第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』。

SQLDescribeCol

```
/*****************************************************************
** file = typical.c
...
/*****************************************************************
** display_results
**
**  - for each column
**      - get column name
**      - bind column
**  - display column headings
**  - fetch each row
**      - if value truncated, build error message
**      - if column null, set value to "NULL"
**      - display row
**      - print truncation message
**  - free local storage
*****************************************************************/
display_results(SQLHSTMT hstmt,
                SQLSMALLINT nresultcols)
{
SQLCHAR         colname[32];
SQLSMALLINT     coltype;
SQLSMALLINT     colnamelen;
SQLSMALLINT     nullable;
SQLINTEGER      collen[MAXCOLS];
SQLSMALLINT     scale;
SQLINTEGER      outlen[MAXCOLS];
SQLCHAR *       data[MAXCOLS];
SQLCHAR         errmsg[256];
SQLRETURN       rc;
SQLINTEGER      i;
SQLINTEGER      displaysize;

    for (i = 0; i < nresultcols; i++)
    {
        SQLDescribeCol (hstmt, i+1, colname, sizeof (colname),
        &colnamelen, &coltype, &collen[i], &scale, &nullable);

        /* get display lenght for column */
        SQLColAttributes (hstmt, i+1, SQL_COLUMN_DISPLAY_SIZE, NULL, 0,
            NULL, &displaysize);

        /* set column length to max of display length, and column name
           length.  Plus one byte for null terminator       */
        collen[i] = max(displaysize, strlen((char *) colname) ) + 1;


        /* allocate memory to bind column                            */
        data[i] = (SQLCHAR *) malloc (collen[i]);

        /* bind columns to program vars, converting all types to CHAR */
        SQLBindCol (hstmt, i+1, SQL_CHAR, data[i], collen[i],
&outlen[i]);
    }
    printf("\n");

    /* display result rows                                           */
    while ((rc = SQLFetch (hstmt)) != SQL_NO_DATA_FOUND)
    {
        errmsg[0] = '\0';
        for (i = 0; i < nresultcols; i++)
        {
            /* Build a truncation message for any columns truncated */
            if (outlen[i] >= collen[i])
            {    sprintf ((char *) errmsg + strlen ((char *) errmsg),
                         "%d chars truncated, col %d\n",
                          outlen[i]-collen[i]+1, i+1);
```

```
            }
            if (outlen[i] == SQL_NULL_DATA)
            else
        } /* for all columns in this row  */

        printf ("\n%s", errmsg);  /* print any truncation messages    */
    } /* while rows to fetch */

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

}/* end display_results
```

参考

- 第 58 页的『SQLColAttributes — 列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 180 页的『SQLNumResultCols — 获取结果列数』
- 第 186 页的『SQLPrepare — 准备语句』

SQLDescribeParam — 返回参数标记的描述

用途

SQLDescribeParam() 返回与已准备的 SQL 语句相关联的参数标记的描述。此信息也存在于实现参数描述符（IPD）的字段中。

语法

```
SQLRETURN   SQLDescribeParam (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       ParameterNumber,
                              SQLSMALLINT       *DataTypePtr,
                              SQLINTEGER        *ParameterSizePtr,
                              SQLSMALLINT       *DecimalDigitsPtr,
                              SQLSMALLINT       *NullablePtr);
```

函数自变量

表 44. SQLDescribeParam 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLSMALLINT	ParameterNumber	输入	参数标记号从 1 开始以递增参数次序按顺序排序。
SQLSMALLINT *	DataTypePtr	输出	指向一个缓冲区的指针，在该缓冲区中返回参数的 SQL 数据类型。
SQLINTEGER *	ParameterSizePtr	输出	指向一个缓冲区的指针，在该缓冲区中返回列的大小或数据源定义的对应参数标记的表达式。
SQLSMALLINT *	DecimalDigitsPtr	输出	指向一个缓冲区的指针，在该缓冲区中返回列的小数位数或数据源定义的对应参数的表达式。
SQLSMALLINT *	NullablePtr	输出	指向一个缓冲区的指针，在该缓冲区中返回指示参数是否允许空值的值。此值是从 IPD 的 SQL_DESC_NULLABLE 字段读取的。 可以是下列其中一项: • SQL_NO_NULLS — 参数不允许空值（这是缺省值）。 • SQL_NULLABLE — 参数允许空值。 • SQL_NULLABLE_UNKNOWN — 不能确定参数是否允许空值。

用法

参数标记从 1 开始按它们在 SQL 语句中的出现次序以递增参数次序编号。

SQLDescribeParam() 不返回 SQL 语句中的参数的类型（输入、输出或输入输出）。除过程调用中的参数之外，SQL 语句中的所有参数都是输入参数。要确定过程调用中的每个参数的类型，应用程序需要调用 SQLProcedureColumns()。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 45. SQLDescribeParam SQLSTATE

SQLSTATE	描述	说明
01000	警告	信息性消息。（函数将返回 SQL_SUCCESS_WITH_INFO。）
07009	描述符索引无效	对自变量 *ParameterNumber* 指定的值小于 1。 对自变量 *ParameterNumber* 指定的值大于相关联的 SQL 语句中的参数数目。 参数标记是非 DML 语句的一部分。 参数标记是 SELECT 列表的一部分。
08S01	通信链路故障	在函数完成处理之前，DB2 UDB CLI 与它连接的数据源之间的通信链路发生故障。
21S01	插入值列表与列列表不匹配	INSERT 语句中的参数数目与该语句中命名的表中的列数不匹配。
HY000	一般错误	
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY008	已将操作取消	
HY009	自变量值无效	自变量 *DataTypePtr*、*ParameterSizePtr*、*DecimalDigitsPtr* 或 *NullablePtr* 是空指针。
HY010	函数顺序错误	在对 *StatementHandle* 调用 `SQLPrepare()` 或 `SQLExecDirect()` 之前调用了此函数。
HY013	意外的内存处理错误	由于未能访问底层内存对象（这可能是因为内存不足而导致的），所以未能处理此函数调用。

限制

无。

参考

- 第 44 页的『SQLBindParam — 将缓冲区绑定到参数标记』
- 第 56 页的『SQLCancel — 取消语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 186 页的『SQLPrepare — 准备语句』

SQLDisconnect — 与数据源断开连接

用途

SQLDisconnect() 关闭与数据库连接句柄相关联的连接。

在调用此函数之后，调用 SQLConnect() 来连接到另一个数据库，或调用 SQLFreeConnect()。

语法

```
SQLRETURN SQLDisconnect (SQLHDBC    hdbc);
```

函数自变量

表 46. SQLDisconnect 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄

用法

如果应用程序在释放所有与连接相关联的语句句柄之前调用 SQLDisconnect，则 DB2 UDB CLI 将在与数据库成功断开连接之后释放它们。

如果返回了 SQL_SUCCESS_WITH_INFO，则意味着即使成功地与数据库断开连接，也仍存在附加的错误或特定于实现的信息。例如:

- 断开连接后，在进行清理时遇到问题，或者，
- 由于发生独立于应用程序的事件（如通信故障）而导致没有当前连接。

在成功调用 SQLDisconnect() 后，应用程序可以重新使用 *hdbc* 来进行另一个 SQLConnect() 请求。

如果 *hdbc* 参与 DUOW 两阶段落实连接，则可能不会立即发生断开连接。实际的断开连接将在为分布式事务发出下一个落实时发生。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 47. SQLDisconnect SQLSTATE

SQLSTATE	描述	说明
01002	断开连接错误	断开连接期间出错。但是，已成功地断开连接。（函数将返回 SQL_SUCCESS_WITH_INFO。）
08003	连接未打开	自变量 *hdbc* 中指定的连接尚未打开。
25000	事务状态无效	有一个事务正在自变量 *hdbc* 指定的连接上进行处理。该事务将保持活动状态，并且不能将该连接断开连接。

表 47. SQLDisconnect SQLSTATE (续)

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY013 *****	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

请参考 `SQLAllocEnv()`（第 28 页的『示例』）。

参考

- 第 24 页的『SQLAllocConnect — 分配连接句柄』
- 第 68 页的『SQLConnect — 连接到数据源』
- 第 239 页的『SQLTransact — 事务管理』

SQLDriverConnect —（扩充）连接到数据源

用途

SQLDriverConnect() 是 SQLConnect() 的备用函数。这两个函数都建立与目标数据库的连接，但是 SQLDriverConnect() 使用连接字符串来确定数据源名、用户标识和密码。这两个函数是相同的；支持它们的目的都是为了提高兼容性。

语法

```
SQLRETURN   SQLDriverConnect (SQLHDBC           ConnectionHandle,
                              SQLHWND           WindowHandle,
                              SQLCHAR           *InConnectionString,
                              SQLSMALLINT       StringLength1,
                              SQLCHAR           *OutConnectionString,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *StringLength2Ptr,
                              SQLSMALLINT       DriverCompletion);
```

函数自变量

表 48. SQLDriverConnect 自变量

数据类型	自变量	用途	描述
SQLHDBC	*ConnectionHandle*	输入	连接句柄
SQLHWND	*hwindow*	输入	窗口句柄（依赖于平台）：在 Windows 上，这是父窗口句柄。在 OS/2 上，这是父 PM 窗口句柄。在 AIX® 上，这是父 “MOTIF 小配件” 窗口句柄。在 iSeries 上，将其忽略。
SQLCHAR *	*InConnectionString*	输入	完整的、部分的或空的（空指针）连接字符串（参见下面的语法和描述）。
SQLSMALLINT	*StringLength1*	输入	*InConnectionString* 的长度。
SQLCHAR *	*OutConnectionString*	输出	指向完整的连接字符串的缓冲区的指针。 如果已成功地建立了连接，则此缓冲区将包含完整的连接字符串。
SQLSMALLINT	*BufferLength*	输入	*OutConnectionString* 指向的缓冲区的最大大小。
SQLSMALLINT *	*StringLength2Ptr*	输出	指向可以在 *OutConnectionString* 缓冲区中返回的字节数的指针。 如果 *StringLength2Ptr* 的值大于或等于 *BufferLength*，则将 *OutConnectionString* 中的完整连接字符串截断成长度为 *BufferLength* - 1 个字节。
SQLSMALLINT	*DriverCompletion*	输入	指示 DB2 UDB CLI 应在何时提示用户输入更多信息。 可能的值包括: • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE_REQUIRED • SQL_DRIVER_NOPROMPT

用法

连接字符串用来传送完成连接所需的一个或多个值。连接字符串的内容和 *DriverCompletion* 的值共同确定应如何建立连接。

连接字符串语法

以上每个关键字都具有属性，各属性列示如下:

DSN 数据源名。数据库的名称或别名名称。如果 *DriverCompletion* 等于 SQL_DRIVER_NOPROMPT，则数据源名是必需的。

UID 授权名（用户标识符）。

PWD 与授权名相对应的密码。如果该用户标识没有密码，则不指定任何内容（PWD=;）。

iSeries 当前没有 DB2 UDB CLI 定义的关键字。

将验证 *DriverCompletion* 的值是否有效，但无论是否有效都会导致相同的行为。将使用连接字符串中包含的信息来尝试建立连接。如果没有足够的信息，则返回 SQL_ERROR。

在建立连接之后，就返回完整的连接字符串。需要为给定的用户标识设置多个与同一数据库的连接的应用程序应存储这个输出连接字符串。这样，在将来的 `SQLDriverConnect()` 调用中可将此字符串用作输入连接字符串值。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA_FOUND
- SQL_INVALID_HANDLE
- SQL_ERROR

错误状态

这里也可能会返回第 68 页的『SQLConnect — 连接到数据源』生成的所有诊断。下表显示了可能会返回的附加诊断。

表 49. SQLDriverConnect SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	缓冲区 *szConnstrOut* 不够大，无法存放整个连接字符串。自变量 *StringLength2Ptr* 包含可以返回的连接字符串的实际长度。（函数将返回 SQL_SUCCESS_WITH_INFO）

表 49. SQLDriverConnect SQLSTATE (续)

SQLSTATE	描述	说明
01S00	连接字符串属性无效	在输入连接字符串中指定了无效的关键字或属性值，但由于发生下列其中一种情况，所以与数据源的连接已成功: • 忽略了不识别的关键字。 • 忽略了无效的属性值，改为使用缺省值。 （函数将返回 SQL_SUCCESS_WITH_INFO）
HY009	自变量值无效	自变量 *InConnectionString*、*OutConnectionString* 或 *StringLength2PTR* 是空指针。 自变量 DriverCompletion 不等于 1。
HY090	字符串或缓冲区长度无效	对 *StringLength1* 指定的值小于 0，但不等于 SQL_NTS。 对 *BufferLength* 指定的值小于 0。
HY110	驱动程序完成无效	对自变量 *fCompletion* 指定的值不等于任何一个有效值。

限制

无。

示例

```
/* From CLI sample drivrcon.c */
/* ... */
/*********************************************************************
**   drv_connect - Prompt for connect options and connect             **
*********************************************************************/

int
drv_connect(SQLHENV henv,
            SQLHDBC * hdbc,
            SQLCHAR con_type)
{
    SQLRETURN       rc;
    SQLCHAR         server[SQL_MAX_DSN_LENGTH + 1];
    SQLCHAR         uid[MAX_UID_LENGTH + 1];
    SQLCHAR         pwd[MAX_PWD_LENGTH + 1];
    SQLCHAR         con_str[255];
    SQLCHAR         buffer[255];
    SQLSMALLINT     outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    /* Allocate a connection handle */
    SQLAllocHandle( SQL_HANDLE_DBC,
                        henv,
                        hdbc
                      );
    CHECK_HANDLE( SQL_HANDLE_DBC, *hdbc, rc);

    sprintf((char *)con_str, "DSN=%s;UID=%s;PWD=%s;",
            server, uid, pwd);
```

```
    rc = SQLDriverConnect(*hdbc,
            (SQLHWND) NULL,
            con_str,
            SQL_NTS,
            buffer, 255, &outlen,
            SQL_DRIVER_NOPROMPT);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database, RC= %ld\n", rc);
        CHECK_HANDLE( SQL_NULL_HENV, *hdbc, rc);
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
        return (SQL_SUCCESS);
    }
}
```

参考

- 第 68 页的『SQLConnect — 连接到数据源』

SQLEndTran — 落实或回滚事务

用途

SQLEndTran() 落实或回滚连接中的当前事务。

将落实或回滚从进行连接时开始或从上次调用 SQLEndTran() 时开始（以较近者为准）在该连接上对数据库执行的所有更改。

如果某个事务正在该连接上活动，则应用程序在可以与数据库断开连接之前必须调用 SQLEndTran()。

语法

```
SQLRETURN SQLEndTran (SQLSMALLINT    hType,
                      SQLINTEGER     handle,
                      SQLSMALLINT    fType);
```

函数自变量

表 50. SQLEndTran 自变量

数据类型	自变量	用途	描述
SQLSMALLINT	*hType*	输入	句柄的类型。它必须包含 SQL_HANDLE_ENV 或 SQL_HANDLE_DBC。
SQLINTEGER	*handle*	输入	执行 COMMIT 或 ROLLBACK 时要使用的句柄。
SQLSMALLINT	*fType*	输入	期望的事务操作。此自变量的值必须是下列其中之一: • SQL_COMMIT • SQL_ROLLBACK • SQL_COMMIT_HOLD • SQL_ROLLBACK_HOLD • SQL_SAVEPOINT_NAME_ROLLBACK • SQL_SAVEPOINT_NAME_RELEASE

用法

使用 SQL_COMMIT 或 SQL_ROLLBACK 来完成事务具有下列效果:

- 在调用 SQLEndTran() 之后，语句句柄仍然有效。
- 在各个事务之间，保留游标名、绑定的参数和列绑定。
- 关闭已打开的游标，废弃任何暂挂检索的结果集。

使用 SQL_COMMIT_HOLD 或 SQL_ROLLBACK_HOLD 完成事务仍然会落实或回滚数据库更改，但不会导致关闭游标。

如果该连接上当前没有活动事务，则调用 SQLEndTran() 对数据库服务器没有影响，并返回 SQL_SUCCESS。

在执行 COMMIT 或 ROLLBACK 时，SQLEndTran() 可能会因为丢失连接而失败。在这种情况下，应用程序可能无法确定是否已处理了 COMMIT 或 ROLLBACK，因此您可能需要数据库管理员的帮助。有关事务作业记录和其它事务管理任务的更多信息，请参考 DBMS 产品信息。

当使用 SQL_SAVEPOINT_NAME_ROLLBACK 或 SQL_SAVEPOINT_NAME_RELEASE 时，必须已使用 SQLSetConnectAttr 来设置保存点名。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 51. SQLEndTran SQLSTATE

SQLSTATE	描述	说明
08003	连接未打开	*hdbc* 未处于已连接状态。
08007	在事务进行期间，连接发生故障。	在函数执行期间，与 *hdbc* 相关联的连接发生故障，并且无法确定在发生故障之前是否已执行所请求的 COMMIT 或 ROLLBACK。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	使用了 SQL_SAVEPOINT_NAME_ROLLBACK 或 SQL_SAVEPOINT_NAME_RELEASE，但没有通过对属性 SQL_ATTR_SAVEPOINT_NAME 调用 SQLSetConnectAttr() 来建立保存点名。
HY012	事务操作状态无效	对自变量 *fType* 指定的值既不是 SQL_COMMIT 也不是 SQL_ROLLBACK。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

SQLError — 检索错误信息

用途

SQLError() 返回与最近对特定语句、连接句柄和环境句柄调用的 DB2 UDB CLI 函数相关联的诊断信息。

此信息由标准化的 SQLSTATE、本机错误代码和文本消息组成。有关更多信息，请参考第 15 页的『DB2 UDB CLI 应用程序中的诊断』。

在从另一个函数调用接收到返回码 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 之后，请调用 SQLError()。

语法

```
SQLRETURN SQLError (SQLHENV       henv,
                    SQLHDBC       hdbc,
                    SQLHSTMT      hstmt,
                    SQLCHAR       *szSqlState,
                    SQLINTEGER    *pfNativeError,
                    SQLCHAR       *szErrorMsg,
                    SQLSMALLINT   cbErrorMsgMax,
                    SQLSMALLINT   *pcbErrorMsg);
```

函数自变量

表 52. SQLError 自变量

数据类型	自变量	用途	描述
SQLHENV	*henv*	输入	环境句柄。要获取与环境相关联的诊断信息，请传送有效的环境句柄。请分别将 *hdbc* 和 *hstmt* 设置为 SQL_NULL_HDBC 和 SQL_NULL_HSTMT。
SQLHDBC	*hdbc*	输入	数据库连接句柄。要获取与连接相关联的诊断信息，请传送有效的数据库连接句柄，并将 *hstmt* 设置为 SQL_NULL_HSTMT。将忽略 *henv* 自变量。
SQLHSTMT	*hstmt*	输入	语句句柄。要获取与语句相关联的诊断信息，请传送有效的语句句柄。将忽略 *henv* 和 *hdbc* 自变量。
SQLCHAR *	*szSqlState*	输出	SQLSTATE，作为 5 个字符的字符串返回，并且以空字符终止。前两个字符指示错误类；跟着的三个字符指示子类。这些值直接与 X/Open SQL CAE 规范和 ODBC 规范中定义的 SQLSTATE 值相对应，而与特定于 IBM 的和特定于产品的 SQLSTATE 值有矛盾。
SQLINTEGER *	*pfNativeError*	输出	本机错误代码。在 DB2 UDB CLI 中，*pfNativeError* 自变量包含 DBMS 返回的 SQLCODE 值。如果错误是由 DB2 UDB CLI 而不是由 DBMS 生成的，则将此字段设置为 -99999。
SQLCHAR *	*szErrorMsg*	输出	指向一个缓冲区的指针，该缓冲区包含由实现定义的消息文本。在 DB2 UDB CLI 中，只返回 DBMS 生成的消息；DB2 UDB CLI 本身不返回任何描述问题的消息文本。

表 52. SQLError 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT	*cbErrorMsgMax*	输入	缓冲区 *szErrorMsg* 的最大长度（即分配的长度）。建议的分配长度是 SQL_MAX_MESSAGE_LENGTH + 1。
SQLSMALLINT *	*pcbErrorMsg*	输出	指向可以返回到 *szErrorMsg* 缓冲区的总字节数的指针。

用法

这些 SQLSTATE 是 X/OPEN SQL CAE 和 X/Open SQL CLI 快照定义的那些 SQLSTATE，而与特定于 IBM 的和特定于产品的 SQLSTATE 值有矛盾。

获取诊断信息时要执行的操作如下:

- 要获取与环境相关联的诊断信息，请传送有效的环境句柄。请分别将 *hdbc* 和 *hstmt* 设置为 SQL_NULL_HDBC 和 SQL_NULL_HSTMT。
- 要获取与连接相关联的诊断信息，请传送有效的数据库连接句柄，并将 *hstmt* 设置为 SQL_NULL_HSTMT。将忽略 *henv* 自变量。
- 要获取与语句相关联的诊断信息，请传送有效的语句句柄。将忽略 *henv* 和 *hdbc* 自变量。

在使用同一个句柄调用除 SQLError() 之外的函数之前，如果没有检索由一个 DB2 UDB CLI 函数生成的诊断信息，则关于前一个函数调用的信息将丢失。无论是否对第二个 DB2 UDB CLI 函数调用生成了诊断信息，情况均如此。

为了避免截断错误消息，请声明长度为 SQL_MAX_MESSAGE_LENGTH + 1 的缓冲区。消息文本永远不会超出这个长度。

返回码

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND
- SQL_SUCCESS

诊断

由于 SQLError() 不为它自己生成诊断信息，所以没有定义 SQLSTATE。如果自变量 szSqlState、pfNativeError、szErrorMsg 或 pcbErrorMsg 是空指针，则返回 SQL_ERROR。

示例

有关以下示例的完整列表，请参考第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』。

```
/************************************************************************
** file = typical.c
************************************************************************/
int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
```

SQLError

```
SQLINTEGER  sqlcode;
SQLSMALLINT length;


    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR *****\n");
        printf("         SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };
    return (0);

}
```

SQLExecDirect — 直接执行语句

用途

SQLExecDirect 直接执行指定的 SQL 语句。语句只能执行一次。并且，连接的数据库服务器必须能够准备该语句。

语法

```
SQLRETURN SQLExecDirect (SQLHSTMT      hstmt,
                         SQLCHAR       *szSqlStr,
                         SQLINTEGER    cbSqlStr);
```

函数自变量

表 53. SQLExecDirect 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄。一定不能有打开的游标与 *hstmt* 相关联，有关更多信息，参见第 114 页的『SQLFreeStmt — 释放（或重设）语句句柄』。
SQLCHAR *	*szSqlStr*	输入	SQL 语句字符串。连接的数据库服务器必须能够准备该语句。
SQLINTEGER	*cbSqlStr*	输入	*szSqlStr* 自变量的内容的长度。必须将长度设置为语句的精确长度，或者，如果语句以空终止的话，则设置为 SQL_NTS。

用法

SQL 语句不能是 COMMIT 或 ROLLBACK。相反，必须通过调用 SQLTransact() 来发出 COMMIT 或 ROLLBACK。有关受支持的 SQL 语句的更多信息，请参考第 4 页的表 1。

SQL 语句字符串可包含参数标记。参数标记由“?”字符表示，它用来指示语句中的一个位置，在调用 SQLExecDirect() 时，将在该位置替代应用程序变量的值。SQLBindParam() 将应用程序变量绑定到每个参数标记（使应用程序变量与参数标记相关联），以指示在传送数据时是否应该执行任何数据转换。在调用 SQLExecDirect() 之前，必须绑定所有参数。

如果 SQL 语句是 SELECT，则 SQLExecDirect() 生成游标名并打开游标。如果应用程序已使用 SQLSetCursorName() 来将游标名与语句句柄相关联，则 DB2 UDB CLI 将应用程序生成的游标名与内部生成的游标名相关联。

要从 SELECT 语句生成的结果集检索行，请在 SQLExecDirect() 成功返回后调用 SQLFetch()。

如果 SQL 语句是定位型 DELETE 或定位型 UPDATE，则该语句引用的游标必须定位在某一行上。另外，必须在同一个连接句柄下的单独语句句柄上定义 SQL 语句。

该语句句柄上一定不能有打开的游标。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果 SQL 语句是搜索型 UPDATE 或搜索型 DELETE，并且没有满足搜索条件的行，则返回 SQL_NO_DATA_FOUND。

诊断

表 54. SQLExecDirect SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *szSqlStr* 是空指针。 自变量 *cbSqlStr* 小于 1，但不等于 SQL_NTS。
HY010	函数顺序错误	没有连接，或者对此语句句柄没有已打开的游标。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

注: 在执行语句时，DBMS 可能生成了许多其它 SQLSTATE 值。

示例

请参考 `SQLFetch()`（第 98 页的『示例』）。

参考

- 第 93 页的『SQLExecute — 执行语句』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 222 页的『SQLSetParam — 设置参数』

SQLExecute — 执行语句

用途

SQLExecute() 执行使用 SQLPrepare() 成功准备的语句，并且执行一次或多次。将使用任何已通过 SQLBindParam() 绑定到参数标记的应用程序变量的当前值来执行语句。

语法

```
SQLRETURN SQLExecute (SQLHSTMT     hstmt);
```

函数自变量

表 55. SQLExecute 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄。一定不能有打开的游标与 hstmt 相关联，有关更多信息，参见第 114 页的『SQLFreeStmt — 释放（或重设）语句句柄』。

用法

SQL 语句字符串可包含参数标记。参数标记由“?”字符表示，它用来指示语句中的一个位置，在调用 SQLExecute() 时，将在该位置替代应用程序变量的值。SQLBindParam() 用来将应用程序变量绑定到每个参数标记（使应用程序变量与参数标记相关联），以及指示在传送数据时是否应该执行任何数据转换。在调用 SQLExecute() 之前，必须绑定所有参数。

在应用程序处理了来自 SQLExecute() 调用的结果之后，它可以使用应用程序变量中的新值（或相同的那些值）来再次执行语句。

不能通过调用 SQLExecute() 来重新执行 SQLExecDirect() 执行的语句；必须首先调用 SQLPrepare()。

如果准备的 SQL 语句是 SELECT，则 SQLExecute() 生成游标名并打开游标。如果应用程序已使用 SQLSetCursorName() 来将游标名与语句句柄相关联，则 DB2 UDB CLI 将应用程序生成的游标名与内部生成的游标名相关联。

要将 SELECT 语句执行多次，则应用程序必须通过使用 SQL_CLOSE 选项调用 SQLFreeStmt() 来关闭游标。在调用 SQLExecute() 时，该语句句柄上一定不能有打开的游标。

要从 SELECT 语句生成的结果集检索行，请在 SQLExecute() 成功返回后调用 SQLFetch()。

如果 SQL 语句是定位型 DELETE 或定位型 UPDATE，则在调用 SQLExecute() 时，该语句引用的游标必须定位在某一行上，并且必须在同一个连接句柄下的单独语句句柄上定义该游标。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLExecute

- SQL_NO_DATA_FOUND

如果 SQL 语句是搜索型 UPDATE 或搜索型 DELETE，并且没有满足搜索条件的行，则返回 SQL_NO_DATA_FOUND。

诊断

SQLExecute() 的 SQLSTATE 包括 SQLExecDirect() 的除 **HY**009 之外的所有那些 SQLSTATE（参考第 92 页的表 54）以及下表中的 SQLSTATE。

表 56. SQLExecute SQLSTATE

SQLSTATE	描述	说明
HY010	函数顺序错误	指定的 *hstmt* 未处于已准备状态。在没有首先调用 SQLPrepare 的情况下调用了 SQLExecute()。

注: 在执行语句时，DBMS 可能生成了许多其它 SQLSTATE 值。

示例

请参考 SQLPrepare()（第 187 页的『示例』）

参考

- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 186 页的『SQLPrepare — 准备语句』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 222 页的『SQLSetParam — 设置参数』

SQLExtendedFetch — 取装行数组

用途

SQLExtendedFetch() 通过以数组形式对每个绑定列返回包含多行（称为行集）的数据块来扩展 SQLFetch() 的功能。SQLSetStmtAttr() 调用上的 SQL_ROWSET_SIZE 属性确定了行集的大小。

要每次取装一行数据，应用程序应调用 SQLFetch()。

语法

```
SQLRETURN   SQLExtendedFetch (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       FetchOrientation,
                              SQLINTEGER        FetchOffset,
                              SQLINTEGER        *RowCountPtr,
                              SQLSMALLINT       *RowStatusArray);
```

函数自变量

表 57. SQLExtendedFetch 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLSMALLINT	FetchOrientation	输入	取装方向。请查看第 103 页的表 62 以了解可能的值。
SQLINTEGER	FetchOffset	输入	用于相对定位的行偏移。
SQLINTEGER *	RowCountPtr	输出	实际取装的行数。如果处理期间发生错误，则 *RowCountPtr* 指向（在行集中）位于发生错误的那一行之前的行的序数位置。如果检索第一行时发生错误，则 *RowCountPtr* 指向值 0。
SQLSMALLINT *	RowStatusArray	输出	状态值的数组。元素数目必须与行集中的行数（由 SQL_ROWSET_SIZE 属性定义）相等。将返回所取装的每一行的状态值: • SQL_ROW_SUCCESS 如果取装的行数小于状态数组中的元素数目（即小于行集的大小），则将其余的状态元素设置为 SQL_ROW_NOROW。 DB2 UDB CLI 无法检测在开始取装之后是否已更新或删除了某一行。因此，不报告下列由 ODBC 定义的状态值: • SQL_ROW_DELETED • SQL_ROW_UPDATED

用法

SQLExtendedFetch() 用来对一组行执行数组取装。应用程序通过使用 SQL_ROWSET_SIZE 属性调用 SQLSetStmtAttr() 来指定数组的大小。

在第一次调用 SQLExtendedFetch() 之前，游标定位在第一行之前。在调用 SQLExtendedFetch() 之后，游标定位在结果集中与刚刚检索的行集中的最后一个行元素相对应的行上。

SQLExtendedFetch

对于结果集中的任何已通过 SQLBindCol() 函数绑定的列，DB2 UDB CLI 根据需要转换绑定列的数据并将其存储在与这些列绑定的位置中。必须以行方向的方式绑定结果集。这表示第一行的所有列的值将是连续的，后面跟着第二行的值，依此类推。并且，如果使用指示符变量，则它们全都将在一个连续的存储位置中返回。

当使用此过程来检索多行时，必须将所有列绑定，并且存储器必须是连续的。当使用此函数来从 SQL 过程结果集检索行时，只支持 SQL_FETCH_NEXT 方向。用户负责为 SQL_ROWSET_SIZE 中指定的行数分配足够的存储器。

为了使 SQLExtendedFetch() 能够使用任何除 SQL_FETCH_NEXT 之外的方向，游标必须是可滚动游标。有关设置 SQL_ATTR_CURSOR_SCROLLABLE 属性的信息，参见第 223 页的『SQLSetStmtAttr — 设置语句属性』。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

错误状态

表 58. SQLExtendedFetch SQLSTATE

SQLSTATE	描述	说明
HY009	自变量值无效	自变量值 *RowCountPtr* 或 *RowStatusArray* 是空指针。 不识别对自变量 FetchOrientation 指定的值。
HY010	函数顺序错误	在调用 SQLFetch() 之后并在使用 SQL_CLOSE 选项调用 SQLFreeStmt() 之前对 StatementHandle 调用了 SQLExtendedFetch()。 在对 *StatementHandle* 调用 SQLPrepare() 或 SQLExecDirect() 之前调用了此函数。 在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。

限制

无。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 93 页的『SQLExecute — 执行语句』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 97 页的『SQLFetch — 取装下一行』

SQLFetch — 取装下一行

用途

SQLFetch() 使游标前进到结果集的下一行，并检索任何已绑定的列。

SQLFetch() 可用来将数据直接接收到您使用 SQLBindCol() 指定的变量中，也可以在取装之后通过调用 SQLGetData() 来分别接收列。如果在绑定列时指示了转换，则调用 SQLFetch() 时也将执行数据转换。

语法

```
SQLRETURN SQLFetch (SQLHSTMT    hstmt);
```

函数自变量

表 59. SQLFetch 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄

用法

仅当最近对 *hstmt* 执行的语句是 SELECT 时，才可以调用 SQLFetch()。

使用 SQLBindCol() 绑定的应用程序变量的数目一定不能超出结果集中的列数，否则 SQLFetch() 将失败。

如果尚未调用 SQLBindCol() 来绑定任何列，则 SQLFetch() 不返回数据给应用程序，而仅仅使游标前进。在这种情况下，可接着调用 SQLGetData() 来个别地获取所有的列。当 SQLFetch() 使游标前进到下一行时，将废弃未绑定的列中的数据。

如果任何绑定的变量不够大，从而无法存放 SQLFetch() 返回的数据，则将数据截断。如果截断了字符数据，则将返回 SQL_SUCCESS_WITH_INFO，并且生成指示已进行截断的 SQLSTATE。SQLBindCol() 延迟输出自变量 *pcbValue* 包含从服务器检索的列数据的实际长度。应用程序应该将输出长度与输入长度（来自 SQLBindCol() 的 *pcbValue* 和 *cbValueMax* 自变量）作比较以确定已将哪些字符列截断。

如果截断涉及小数点右边的位，则不报告数字数据类型的截断。如果截断发生在小数点的左边，则返回错误（请参考诊断部分）。

将图形数据类型的截断视为与字符数据类型的截断相同。只有一处不同，即填充 *rgbValue* 缓冲区，使其成为仍小于或等于 SQLBindCol() 中指定的 *cbValueMax* 的最接近的双字节倍数。在 DB2 UDB CLI 与应用程序之间传送的图形数据永远不会以空终止。

在从结果集检索所有行之后，或者在不需要其余的行时，应调用 SQLFreeStmt() 来关闭游标并废弃其余的数据和相关联的资源。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLFetch

- SQL_NO_DATA_FOUND

如果结果集中没有行，或者先前的 SQLFetch() 调用已从结果集中取装了所有的行，则将返回 SQL_NO_DATA_FOUND。

诊断

表 60. SQLFetch SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	为一列或多列返回的数据已被截断。将在右边对字符串值进行截断。（如果没有发生错误，则返回 SQL_SUCCESS_WITH_INFO。）
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	指定的 *hstmt* 未处于已执行状态。在没有首先调用 SQLExecute 或 SQLExecDirect 的情况下调用了此函数。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

有关代码示例的信息，参见第 viii 页的『代码不保证声明信息』。

```
/*****************************************************************************
** file = fetch.c
**
** Example of executing an SQL statement.
** SQLBindCol & SQLFetch is used to retrive data from the result set
** directly into application storage.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLExecDirect
**        SQLError
**
*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);
```

```
int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc);

/*****************************************************************
** main
** - initialize
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));


    {SQLHSTMT   hstmt;
     SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = 'Eastern'";
     SQLCHAR    deptname[15],
                location[14];
     SQLINTEGER rlength;

        rc = SQLAllocStmt(hdbc, &hstmt);
        if (rc != SQL_SUCCESS )
           check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        rc = SQLBindCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);
        rc = SQLBindCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        printf("Departments in Eastern division:\n");
        printf("DEPTNAME       Location\n");
        printf("-------------- -------------\n");

        while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
        {
            printf("%-14.14s %-13.13s \n", deptname, location);
        }
        if (rc != SQL_NO_DATA_FOUND )
           check_error (henv, hdbc, hstmt, rc);

        rc = SQLFreeStmt(hstmt, SQL_DROP);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
    }


    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
```

```
}/* end main */

/*****************************************************************
** initialize
**  - allocate environment handle
**  - allocate connection handle
**  - prompt for server, user id, & password
**  - connect to server
*****************************************************************/

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc)
{
SQLCHAR     server[SQL_MAX_DSN_LENGTH],
            uid[30],
            pwd[30];
SQLRETURN   rc;

    rc = SQLAllocEnv (henv);         /* allocate an environment handle    */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    rc = SQLAllocConnect (*henv, hdbc);  /* allocate a connection handle   */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
    else
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }

    return(SQL_SUCCESS);
}/* end initialize */

/*****************************************************************
** terminate
**  - disconnect
**  - free connection handle
**  - free environment handle
*****************************************************************/
int terminate(SQLHENV henv,
              SQLHDBC hdbc)
{
SQLRETURN   rc;

    rc = SQLDisconnect (hdbc);               /* disconnect from database  */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeConnect (hdbc);              /* free connection handle    */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeEnv (henv);                  /* free environment handle   */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
```

```
    return(rc);
}/* end terminate */

/*****************************************************************
**  - print_error   - call SQLError(), display SQLSTATE and message
*****************************************************************/

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER  sqlcode;
SQLSMALLINT length;


    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR *****\n");
        printf("         SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };

    return ( SQL_ERROR);
} /* end print_error */

/*****************************************************************
**  - check_error   - call print_error(), checks severity of return code
*****************************************************************/
int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
        else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
```

```
        break;
    }
    return(SQL_SUCCESS);

} /* end check_error */
```

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 93 页的『SQLExecute — 执行语句』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 116 页的『SQLGetCol — 检索结果集的某一行的其中一列』
- 第 103 页的『SQLFetchScroll — 从可滚动游标取装』

SQLFetchScroll — 从可滚动游标取装

用途

SQLFetchScroll() 根据所请求的方向定位游标，然后检索任何已绑定的列。

SQLFetchScroll() 可用来将数据直接接收到您使用 SQLBindCol() 指定的变量中，也可以在取装之后通过调用 SQLGetData() 来分别接收列。如果在绑定列时指示了数据转换，则调用 SQLFetchScroll() 时也将执行转换。

语法

```
SQLRETURN SQLFetchScroll (SQLHSTMT    hstmt,
                          SQLSMALLINT fOrient,
                          SQLINTEGER  fOffset);
```

函数自变量

表 61. SQLFetchScroll 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*fOrient*	输入	取装方向。参见表 62 以了解可能的值。
SQLINTEGER	*fOffset*	输入	用于相对定位的行偏移。

用法

仅当最近对 *hstmt* 执行的语句是 SELECT 时，才可以调用 SQLFetchScroll()。

除了 *fOrient* 参数在检索任何数据之前定位游标这一点之外，SQLFetchScroll() 的行为与 SQLFetch() 类似。为了使 SQLFetchScroll() 能够使用任何除 SQL_FETCH_NEXT 之外的方向，游标必须是可滚动游标。有关设置 SQL_ATTR_CURSOR_SCROLLABLE 属性的信息，参见第 223 页的『SQLSetStmtAttr — 设置语句属性』。

当使用此函数来从 SQL 过程结果集检索行时，只支持 SQL_FETCH_NEXT 方向。

表 62. 语句属性

fOrient	描述
SQL_FETCH_NEXT	移至位于当前游标位置之后的行。
SQL_FETCH_FIRST	移至结果集的第一行。
SQL_FETCH_LAST	移至结果集的最后一行。
SQL_FETCH_PRIOR	移至位于当前游标位置之前的行。
SQL_FETCH_RELATIVE	如果 *fOffset* 是: • 正数，游标前进该行数。 • 负数，游标后退该行数。 • 零，不移动游标。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

SQLFetchScroll

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

诊断

表 63. SQLFetchScroll SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	为一列或多列返回的数据已被截断。将在右边对字符串值进行截断。（如果没有发生错误，则返回 SQL_SUCCESS_WITH_INFO。）
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	方向无效。
HY010	函数顺序错误	指定的 *hstmt* 未处于已执行状态。在没有首先调用 SQLExecute 或 SQLExecDirect 的情况下调用了此函数。
HY013 *****	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 93 页的『SQLExecute — 执行语句』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 116 页的『SQLGetCol — 检索结果集的某一行的其中一列』
- 第 97 页的『SQLFetch — 取装下一行』

SQLForeignKeys — 获取外键列的列表

用途

SQLForeignKeys() 返回关于所指定的表的外键的信息。将在 SQL 结果集中返回此信息，可以使用那些用来检索由查询生成的结果的函数来处理此结果集。

语法

```
SQLRETURN   SQLForeignKeys   (SQLHSTMT          StatementHandle,
                              SQLCHAR           *PKCatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *PKSchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *PKTableName,
                              SQLSMALLINT       NameLength3,
                              SQLCHAR           *FKCatalogName,
                              SQLSMALLINT       NameLength4,
                              SQLCHAR           *FKSchemaName,
                              SQLSMALLINT       NameLength5,
                              SQLCHAR           *FKTableName,
                              SQLSMALLINT       NameLength6);
```

函数自变量

表 64. SQLForeignKeys 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLCHAR *	PKCatalogName	输入	主键表的目录限定符。这必须是空指针或长度为零的字符串。
SQLSMALLINT	NameLength1	输入	*PKCatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	PKSchemaName	输入	主键表的模式限定符。
SQLSMALLINT	NameLength2	输入	*PKSchemaName* 的长度
SQLCHAR *	PKTableName	输入	包含主键的表名的名称。
SQLSMALLINT	NameLength3	输入	*PKTableName* 的长度。
SQLCHAR *	FKCatalogName	输入	包含外键的表的目录限定符。这必须是空指针或长度为零的字符串。
SQLSMALLINT	NameLength4	输入	*FKCatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	FKSchemaName	输入	包含外键的表的模式限定符。
SQLSMALLINT	NameLength5	输入	*FKSchemaName* 的长度。
SQLCHAR *	FKTableName	输入	包含外键的表的名称。
SQLSMALLINT	NameLength6	输入	*FKTableName* 的长度。

用法

如果 *PKTableName* 包含表名，并且 *FKTableName* 是空字符串，则 SQLForeignKeys() 将返回一个结果集，该结果集包含所指定的表的主键以及（其它表中）所有引用该表的外键。

如果 *FKTableName* 包含表名，并且 *PKTableName* 是空字符串，则 SQLForeignKeys() 将返回一个结果集，该结果集包含所指定的表中的所有外键以及它们所引用的（其它表中的）主键。

SQLForeignKeys

如果 *PKTableName* 和 *FKTableName* 都包含表名，则 SQLForeignKeys() 将返回 *FKTableName* 中指定的表中的特定外键，那些外键引用 *PKTableName* 中指定的表的主键。这最多只应该是一个键。

如果未指定与表名相关联的模式限定符自变量，则模式名缺省值将是当前对当前连接起作用的那个模式名。

表 65 列示了 SQLForeignKeys() 调用生成的结果集的列。如果请求了与主键相关联的外键，则结果集将按 FKTABLE_CAT、FKTABLE_SCHEM、FKTABLE_NAME 和 ORDINAL_POSITION 进行排序。如果请求了与外键相关联的主键，则结果集将按 PKTABLE_CAT、PKTABLE_SCHEM、PKTABLE_NAME 和 ORDINAL_POSITION 进行排序。

虽然在将来的发行版中可能会添加新列，并可能会更改现有列的名称，但当前列的位置不会更改。

表 65. SQLForeignKeys 返回的列

列号 / 名称	数据类型	描述
1 PKTABLE_CAT	VARCHAR(128)	当前服务器。
2 PKTABLE_SCHEM	VARCHAR(128)	包含 PKTABLE_NAME 的模式的名称。
3 PKTABLE_NAME	VARCHAR(128) 非空	包含主键的表的名称。
4 PKCOLUMN_NAME	VARCHAR(128) 非空	主键列名。
5 FKTABLE_CAT	VARCHAR(128)	当前服务器。
6 FKTABLE_SCHEM	VARCHAR(128)	包含 FKTABLE_NAME 的模式的名称。
7 FKTABLE_NAME	VARCHAR(128) 非空	包含外键的表的名称。
8 FKCOLUMN_NAME	VARCHAR(128) 非空	外键列名。
9 ORDINAL_POSITION	SMALLINT 非空	列在键中的序数位置，从 1 开始。
10 UPDATE_RULE	SMALLINT	当 SQL 操作是 UPDATE 时要对外键应用的操作: • SQL_RESTRICT • SQL_NO_ACTION IBM DB2 DBMS 的更新规则始终为 RESTRICT 或 SQL_NO_ACTION。然而，在连接到非 IBM RDBMS 时，ODBC 应用程序可能会遇到下列 UPDATE_RULE 值: • SQL_CASCADE • SQL_SET_NULL
11 DELETE_RULE	SMALLINT	当 SQL 操作是 DELETE 时要对外键应用的操作: • SQL_CASCADE • SQL_NO_ACTION • SQL_RESTRICT • SQL_SET_DEFAULT • SQL_SET_NULL
12 FK_NAME	VARCHAR(128)	外键标识符。如果不适用于数据源，则为空。
13 PK_NAME	VARCHAR(128)	主键标识符。如果不适用于数据源，则为空。

注: DB2 UDB CLI 使用的列名遵循 X/Open CLI CAE 规范样式。列类型、内容和次序与 ODBC 中对 SQLForeignKeys() 结果集定义的那些列类型、内容和次序完全相同。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 66. SQLForeignKeys SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	已在语句句柄上打开了游标。
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *PKTableName* 和 *FKTableName* 都是空指针。
HY010	函数顺序错误	
HY014	没有更多的句柄	由于内部资源不足，所以 DB2 UDB CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。 表的长度或所有者名称的长度大于服务器支持的最大长度。请参考第 143 页的『SQLGetInfo — 获取一般信息』。
HYC00	驱动程序不具有能力	DB2 UDB CLI 不支持将 *catalog* 用作表名的限定符。
HYT00	超时到期	

限制

无。

示例

```
/* From CLI sample browser.c */
/* ... */
SQLRETURN list_foreign_keys( SQLHANDLE hstmt,
                             SQLCHAR * schema,
                             SQLCHAR * tablename
                           ) {

/* ... */
    rc = SQLForeignKeys(hstmt, NULL, 0,
                        schema, SQL_NTS, tablename, SQL_NTS,
                        NULL, 0,
                        NULL, SQL_NTS, NULL, SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) pktable_schem.s, 129,
                    &pktable_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) pktable_name.s, 129,
                    &pktable_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) pkcolumn_name.s, 129,
```

```
                &pkcolumn_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 6, SQL_C_CHAR, (SQLPOINTER) fktable_schem.s, 129,
                &fktable_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) fktable_name.s, 129,
                &fktable_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 8, SQL_C_CHAR, (SQLPOINTER) fkcolumn_name.s, 129,
                &fkcolumn_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 10, SQL_C_SHORT, (SQLPOINTER) &update_rule,
                0, &update_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 11, SQL_C_SHORT, (SQLPOINTER) &delete_rule,
                0, &delete_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 12, SQL_C_CHAR, (SQLPOINTER) fkey_name.s, 129,
                &fkey_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 13, SQL_C_CHAR, (SQLPOINTER) pkey_name.s, 129,
                &pkey_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    printf("Primary Key and Foreign Keys for %s.%s\n", schema, tablename);
    /* Fetch each row, and display */
    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
        printf("  %s  %s.%s.%s\n      Update Rule ",
               pkcolumn_name.s, fktable_schem.s, fktable_name.s, fkcolumn_name.s);
        if (update_rule == SQL_RESTRICT) {
            printf("RESTRICT ");  /* always for IBM DBMSs */
        } else {
            if (update_rule == SQL_CASCADE) {
               printf("CASCADE ");  /* non-IBM only */
            } else {
               printf("SET NULL ");
            }
        }
        printf(", Delete Rule: ");
        if (delete_rule== SQL_RESTRICT) {
            printf("RESTRICT ");  /* always for IBM DBMSs */
        } else {
            if (delete_rule == SQL_CASCADE) {
               printf("CASCADE ");  /* non-IBM only */
            } else {
               if (delete_rule == SQL_NO_ACTION) {
                  printf("NO ACTION ");  /* non-IBM only */
               } else {
                    printf("SET NULL ");
                 }
            }
        }
        printf("\n");
        if (pkey_name.ind > 0 ) {
            printf("     Primary Key Name: %s\n", pkey_name.s);
        }
        if (fkey_name.ind > 0 ) {
            printf("     Foreign Key Name: %s\n", fkey_name.s);
        }
    }
```

参考

- 第 190 页的『SQLPrimaryKeys — 获取表的主键列』
- 第 231 页的『SQLStatistics — 获取基本表的索引和统计信息』

SQLFreeConnect — 释放连接句柄

用途

SQLFreeConnect() 使连接句柄无效并将其释放。将释放与该连接句柄相关联的所有 DB2 UDB CLI 资源。

在调用此函数之前，必须调用 SQLDisconnect()。

接着调用 SQLFreeEnv() 来继续终止应用程序，或调用 SQLAllocHandle() 来分配新的连接句柄。

语法

```
SQLRETURN SQLFreeConnect (SQLHDBC    hdbc);
```

函数自变量

表 67. SQLFreeConnect 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄

用法

如果在仍然存在连接的情况下调用此函数，则将返回 SQL_ERROR，并且连接句柄将保持有效。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 68. SQLFreeConnect SQLSTATE

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	在对 *hdbc* 调用 SQLDisconnect() 之前调用了此函数。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

请参考 SQLAllocEnv()（第 28 页的『示例』）。

参考

- 第 80 页的『SQLDisconnect — 与数据源断开连接』
- 第 111 页的『SQLFreeEnv — 释放环境句柄』

SQLFreeEnv — 释放环境句柄

用途

SQLFreeEnv() 使环境句柄无效并将其释放。将释放与该环境句柄相关联的所有 DB2 UDB CLI 资源。

在调用此函数之前，必须调用 SQLFreeConnect()。

此函数是应用程序在终止之前所需执行的最后一个 DB2 UDB CLI 步骤。

语法

```
SQLRETURN SQLFreeEnv (SQLHENV    henv);
```

函数自变量

表 69. SQLFreeEnv 自变量

数据类型	自变量	用途	描述
SQLHENV	*henv*	输入	环境句柄

用法

如果在仍然存在有效连接句柄的情况下调用此函数，则将返回 SQL_ERROR，并且环境句柄将保持有效。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 70. SQLFreeEnv SQLSTATE

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	存在处于已分配或已连接状态的 *hdbc*。在调用 SQLFreeEnv 之前，请对该 *hdbc* 调用 SQLDisconnect 和 SQLFreeConnect。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

请参考 SQLAllocEnv()（第 28 页的『示例』）。

参考

- 第 110 页的『SQLFreeConnect — 释放连接句柄』

SQLFreeHandle — 释放句柄

用途

SQLFreeHandle() 使句柄无效并将其释放。

语法

```
SQLRETURN SQLFreeHandle (SQLSMALLINT htype,
                         SQLINTEGER handle);
```

函数自变量

表 71. SQLFreeHandle 自变量

数据类型	自变量	用途	描述
SQLSMALLINT	*hType*	输入	句柄类型。必须是 SQL_HANDLE_ENV、SQL_HANDLE_DBC、SQL_HANDLE_STMT 或 SQL_HANDLE_DESC。
SQLINTEGER	*handle*	输入	将要释放的句柄

用法

SQLFreeHandle() 组合了 SQLFreeEnv()、SQLFreeConnect() 和 SQLFreeStmt() 的功能。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 72. SQLFreeHandle SQLSTATE

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	存在处于已分配或已连接状态的 *hdbc*。在调用 SQLFreeHandle 之前，请对该 *hdbc* 调用 SQLDisconnect 和 SQLFreeConnect。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 110 页的『SQLFreeConnect — 释放连接句柄』
- 第 111 页的『SQLFreeEnv — 释放环境句柄』

- 第 114 页的『SQLFreeStmt — 释放（或重设）语句句柄』

SQLFreeStmt — 释放（或重设）语句句柄

用途

SQLFreeStmt() 结束对语句句柄所引用的语句的处理。使用此函数来:

- 关闭游标
- 重设参数
- 取消列与变量的绑定
- 删除语句句柄并释放与语句句柄相关联的 DB2 UDB CLI 资源

请在执行 SQL 语句且处理结果之后调用 SQLFreeStmt()。

语法

```
SQLRETURN SQLFreeStmt (SQLHSTMT       hstmt,
                       SQLSMALLINT    fOption);
```

函数自变量

表 73. SQLFreeStmt 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*fOption*	输入	用于指定语句句柄释放方式的选项。此选项必须具有下列其中一个值: • SQL_CLOSE • SQL_DROP • SQL_UNBIND • SQL_RESET_PARAMS

用法

可使用下列选项来调用 SQLFreeStmt():

- SQL_CLOSE

  关闭与语句句柄（*hstmt*）相关联的游标（如果有的话），并废弃所有暂挂的结果。通过在绑定到 *hstmt* 的应用程序变量（如果有的话）中指定相同的或不同的值，应用程序可以通过调用 SQLExecute() 来重新打开该游标。游标名将保留到删除语句句柄或下次成功调用 SQLSetCursorName() 时为止。如果还没有将任何游标与语句句柄相关联，则此选项不起任何作用（不生成警告或错误）。

- SQL_DROP

  释放与输入语句句柄相关联的 DB2 UDB CLI 资源，并使该句柄无效。将关闭打开的游标（如果有的话）并废弃所有暂挂的结果。

- SQL_UNBIND

  释放由先前对此语句句柄进行的 SQLBindCol() 调用绑定的所有列（应用程序变量或文件引用与结果集列之间的关联将断开）。

- SQL_RESET_PARAMS

  释放由先前对此语句句柄进行的 SQLBindParam() 调用设置的所有参数。应用程序变量或文件引用与该语句句柄的 SQL 语句中的参数标记之间的关联将断开。

要重新使用某个语句句柄来执行另一个语句，并且如果前一个语句:

- 是 SELECT，则必须关闭游标。
- 使用不同的参数数目或使用不同类型的参数，则必须重设参数。
- 使用不同的列绑定数目或使用不同类型的列绑定，则必须将列取消绑定。

另外，可以删除语句句柄并分配新的语句句柄。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

如果将 *fOption* 设置为 SQL_DROP，则不返回 SQL_SUCCESS_WITH_INFO，这是因为调用 `SQLError()` 时将没有可使用的语句句柄。

诊断

表 74. SQLFreeStmt SQLSTATE

SQLSTATE	描述	说明
40003 *****	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	对自变量 *fOption* 指定的值不是 SQL_CLOSE、SQL_DROP、SQL_UNBIND 或 SQL_RESET_PARAMS。

示例

请参考 `SQLFetch()`（第 98 页的『示例』）。

参考

- 第 32 页的『SQLAllocStmt — 分配语句句柄』
- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 110 页的『SQLFreeConnect — 释放连接句柄』
- 第 222 页的『SQLSetParam — 设置参数』

SQLGetCol — 检索结果集的某一行的其中一列

用途

SQLGetCol() 检索结果集的当前行中的单个列的数据。此函数是 SQLBindCol() 的备用函数，该函数导致进行 SQLFetch() 调用时将数据直接传送到应用程序变量中。SQLGetCol() 还用来以分块方式检索基于大字符的数据。

在调用 SQLGetCol() 之前，必须调用 SQLFetch()。

在对每一列调用 SQLGetCol() 之后，调用 SQLFetch() 来检索下一行。

语法

```
SQLRETURN SQLGetCol  (SQLHSTMT       hstmt,
                      SQLSMALLINT    icol,
                      SQLSMALLINT    fCType,
                      SQLPOINTER     rgbValue,
                      SQLINTEGER     cbValueMax,
                      SQLINTEGER     *pcbValue);
```

函数自变量

表 75. SQLGetCol 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*icol*	输入	请求对其进行数据检索的列的编号
SQLSMALLINT	*fCType*	输入	由 *icol* 标识的列的应用程序数据类型。支持下列类型: • SQL_CHAR • SQL_VARCHAR • SQL_NUMERIC • SQL_DECIMAL • SQL_BIGINT • SQL_INTEGER • SQL_SMALLINT • SQL_FLOAT • SQL_REAL • SQL_DOUBLE • SQL_GRAPHIC • SQL_VARGRAPHIC • SQL_DATETIME • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP
SQLPOINTER	*rgbValue*	输出	指向一个缓冲区的指针，将在该缓冲区中存储检索到的列数据。

表 75. SQLGetCol 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER	*cbValueMax*	输入	*rgbValue* 指向的缓冲区的最大大小。如果 *fcType* 是 SQL_DECIMAL 或 SQL_NUMERIC，则 *cbValueMax* 必须实际上是精度和标度。指定这两个值的方法是使用 (精度 * 256) + 标度。这也是使用 SQLColAttributes() 时作为这些数据类型的长度（LENGTH）返回的值。
SQLINTEGER *	*pcbValue*	输出	指向一个值的指针，该值指示了 DB2 UDB CLI 可以在 *rgbValue* 缓冲区中返回的字节数。如果正在以分块方式检索数据，则该值包含仍剩余的字节数，这不包括先前的 SQLGetCol() 调用已获取的任何列数据字节。 如果列的数据值为空，则该值为 SQL_NULL_DATA。如果此指针为空，并且 SQLFetch() 已获取包含空数据的列，则此函数将失败，原因是无法报告这一点。 如果 SQLFetch() 已取装包含图形数据的列，则指向 *pcbValue* 的指针一定不能为空，否则此函数将失败，这是因为没有办法将关于在 *rgbValue* 缓冲区中检索到的数据的长度的信息通知应用程序。

用法

只要 *icol* 的值没有指定已绑定的列，就可将 SQLGetCol() 与 SQLBindCol() 一起用于同一行。一般的步骤是:

1. SQLFetch() — 使游标前进到第一行，检索第一行，传送绑定列的数据。
2. SQLGetCol() — 传送指定的（未绑定）列的数据。
3. 对所需的每一列重复步骤 2。
4. SQLFetch() — 使游标前进到下一行，检索下一行，传送绑定列的数据。
5. 对结果集中的每一行重复步骤 2、3 和 4，或重复这些步骤直到不再需要结果集为止。

如果 C 数据类型（*fCType*）是 SQL_CHAR，或者如果 *fCType* 是 SQL_DEFAULT 并且列类型是 CHAR 或 VARCHAR，则 SQLGetCol() 检索长列。

在每个 SQLGetCol() 调用上，如果可供返回的数据值大于或等于 *cbValueMax*，则会发生截断。与指示数据截断的 SQLSTATE 相伴随的函数返回码 SQL_SUCCESS_WITH_INFO 指示发生了截断。应用程序可使用同一个 *icol* 值来再次调用 SQLGetCol()，以获取同一个未绑定列中位于截断发生位置之后的数据。要获取整个列，应用程序应重复这样的调用，直到此函数返回 SQL_SUCCESS 为止。对 SQLGetCol() 的下一个调用将返回 SQL_NO_DATA_FOUND。

要废弃已检索到的列数据部分，应用程序可以调用 SQLGetCol()，并将 *icol* 设置为您感兴趣的下一个列位置。要废弃整一行的未检索到的数据，应用程序应调用 SQLFetch() 以使游标前进到下一行；或者，如果对结果集中的任何其它数据不感兴趣，则应调用 SQLFreeStmt() 来关闭游标。

fCType 输入自变量确定在将列数据放入 *rgbValue* 所指向的存储区域之前所需进行的数据转换（如果有的话）的类型。

SQLGetCol

除非使用 SQLSetEnvAttr() 来更改 SQL_ATTR_OUTPUT_NTS 属性，或者应用程序正在检索多个块中的数据，否则 rgbValue 中返回的内容总是以空终止。如果应用程序正在检索多个块中的数据，则只将空终止字节添加到数据的最后一部分。

如果截断涉及小数点右边的位，则不报告数字数据类型的截断。如果截断发生在小数点的左边，则返回错误（请参考诊断部分）。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果前一个 `SQLGetCol()` 调用已检索此列的所有数据，则返回 SQL_NO_DATA_FOUND。

如果 `SQLGetCol()` 检索到长度为零的字符串，则返回 SQL_SUCCESS。如果是这种情况，则 *pcbValue* 包含 0，并且 *rgbValue* 包含空终止符。

如果前一个 `SQLFetch()` 调用失败，则由于尚未定义结果，所以不应调用 `SQLGetCol()`。

诊断

表 76. SQLGetCol SQLSTATE

SQLSTATE	描述	说明
07006	受限数据类型属性违例	不能将数据值转换为自变量 *fCType* 指定的 C 数据类型。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *cbValueMax* 的值小于 1，并且自变量 *fCType* 是 SQL_CHAR。 指定的列号无效。 自变量 *rgbValue* 或 *pcbValue* 是空指针。
HY010	函数顺序错误	指定的 *hstmt* 不处于游标已定位状态。在没有首先调用 `SQLFetch()` 的情况下调用了此函数。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。
HYC00	驱动程序不具有能力	驱动程序能够识别但不支持所指定的数据类型的 SQL 数据类型。 驱动程序或数据源无法执行所请求的从 SQL 数据类型到应用程序数据 *fCType* 的转换。

限制

ODBC 要求 *icol* 所指定的列的编号不得小于 `SQLGetCol()` 上次对同一语句句柄上的同一行检索的列的编号。ODBC 也不允许使用 `SQLGetCol()` 来检索位于最后一个绑定列之前的列的数据（如果已将行中的任何列绑定的话）。

DB2 UDB CLI 已放松了这两条规则，它允许按任何次序指定 *icol* 的值，并且允许它的值位于绑定列之前，条件是 *icol* 不指定绑定列。

示例

请参考 SQLFetch()（第 98 页的『示例』）以了解使用绑定列与使用 SQLGetCol() 的比较。

有关以下示例中使用的 check_error、initialize 和 terminate 函数的列表，请参考第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』。

```
/************************************************************************
** file = getcol.c
**
** Example of directly executing an SQL statement.
** Getcol is used to retrieve information from the result set.
** Compare to fetch.c
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLError
**        SQLExecDirect        SQLGetCursor
************************************************************************/

#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc);

/*********************************************************************
** main
** - initialize
** - terminate
*********************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc != SQL_SUCCESS) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt;
```

```
     SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = 'Eastern'";
     SQLCHAR    deptname[15],
                location[14];
     SQLINTEGER rlength;

        rc = SQLAllocStmt(hdbc, &hstmt);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        printf("Departments in Eastern division:\n");
        printf("DEPTNAME       Location\n");
        printf("-------------- -------------\n");

        while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
        {
            rc = SQLGetCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15, &rlength);
            rc = SQLGetCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14, &rlength);
            printf("%-14.14s %-13.13s \n", deptname, location);
        }
        if (rc != SQL_NO_DATA_FOUND )
            check_error (henv, hdbc, hstmt, rc);
    }

    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (SQL_SUCCESS);

}/* end main */
```

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 97 页的『SQLFetch — 取装下一行』

SQLGetConnectAttr — 获取连接属性的值

用途

SQLGetConnectAttr() 返回所指定的连接选项的当前设置。

这些选项是使用 SQLSetConnectAttr() 函数设置的。

语法

```
SQLRETURN SQLGetConnectAttr(   SQLHDBC      hdbc,
                               SQLINTEGER   fAttr,
                               SQLPOINTER   pvParam),;
                               SQLINTEGER   bLen,
                               SQLINTEGER   *sLen);
```

函数自变量

表 77. SQLGetConnectAttr 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄
SQLINTEGER	*fAttr*	输入	要检索的属性。有关更多信息，请参考第 206 页的表 147。
SQLPOINTER	*pvParam*	输出	与 *fAttr* 相关联的值。根据 *fAttr* 的值的不同，这可以是 32 位整数值，也可以是指向以空终止的字符串的指针。
SQLINTEGER	*bLen*	输入	如果属性值是字符串的话，此自变量是要在 pvParm 中存储的最大字节数；否则不使用此自变量。
SQLINTEGER *	*sLen*	输出	如果属性是字符串的话，此自变量是输出数据的长度；否则不使用此自变量。

用法

如果调用 SQLGetConnectAttr()，并且尚未通过 SQLSetConnectAttr 设置指定的 *fAttr*，并且它没有缺省值，则 SQLGetConnectAttr() 将返回 SQL_NO_DATA_FOUND。

不能通过 SQLGetConnectAttr() 来检索语句选项设置。

诊断

表 78. SQLGetConnectAttr SQLSTATE

SQLSTATE	描述	说明
08003	连接未打开	指定了需要已打开的连接的 *fAttr*。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	属性类型超出范围	指定了无效的 *fAttr* 值。 自变量 *pvParam* 是空指针。
HYC00	驱动程序不具有能力	能够识别但不支持 *fAttr*。

SQLGetConnectOption — 返回连接选项的当前设置

用途

SQLGetConnectOption() 返回所指定的连接选项的当前设置。

这些选项是使用 SQLSetConnectOption() 函数设置的。

语法

```
SQLRETURN SQLGetConnectOption( HDBC         hdbc,
                              SQLSMALLINT   fOption,
                              SQLPOINTER    pvParam);
```

函数自变量

表 79. SQLGetConnectOption 自变量

数据类型	自变量	用途	描述
HDBC	*hdbc*	输入	连接句柄
SQLSMALLINT	*fOption*	输入	要检索的选项。有关更多信息，请参考第 206 页的表 147。
SQLPOINTER	*pvParam*	输出	与 *fOption* 相关联的值。根据 *fOption* 的值的不同，这可以是 32 位整数值，也可以是指向以空终止的字符串的指针。所返回的任何字符串的最大长度是 SQL_MAX_OPTION_STRING_LENGTH 字节（不包括空终止字节）。

用法

SQLGetConnectOption() 与 SQLGetConnectAttr() 提供相同的功能，支持这两个函数的目的都是为了提高兼容性。

如果调用 SQLGetConnectOption()，并且尚未通过 SQLSetConnectOption 设置指定的 *fOption*，并且它没有缺省值，则 SQLGetConnectOption() 将返回 SQL_NO_DATA_FOUND。

不能通过 SQLGetConnectOption() 来检索语句选项设置。

诊断

表 80. SQLGetConnectOption SQLSTATE

SQLSTATE	描述	说明
08003	连接未打开	指定了需要已打开的连接的 *fOption*。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	选项类型超出范围	指定了无效的 *fOption* 值。 自变量 *pvParam* 是空指针。
HYC00	驱动程序不具有能力	能够识别但不支持 *fOption*。

SQLGetCursorName — 获取游标名

用途

SQLGetCursorName() 返回与输入语句句柄相关联的游标名。如果已通过调用 SQLSetCursorName() 来显式地设置游标名，则返回此名称，否则返回隐式生成的名称。

语法

```
SQLRETURN SQLGetCursorName (SQLHSTMT       hstmt,
                            SQLCHAR        *szCursor,
                            SQLSMALLINT    cbCursorMax,
                            SQLSMALLINT    *pcbCursor);
```

函数自变量

表 81. SQLGetCursorName 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLCHAR *	*szCursor*	输出	游标名
SQLSMALLINT	*cbCursorMax*	输入	缓冲区 *szCursor* 的长度
SQLSMALLINT *	*pcbCursor*	输出	可以对 *szCursor* 返回的字节数

用法

如果已使用 SQLSetCursorName() 来设置名称，或者已对语句句柄执行 SELECT 语句，则 SQLGetCursorName() 将返回游标名。如果还没有发生过这两种情况，则调用 SQLGetCusorName() 将导致错误。

如果使用 SQLSetCursorName() 显式地设置了名称，则在删除语句之前，或在设置另一个显式名称之前，将返回此名称。

如果未设置显式名称，则执行 SELECT 语句时将生成隐式名称，并且将返回此名称。隐式游标名始终以 SQLCUR 开头。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 82. SQLGetCursorName SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	*szCursor* 中返回的游标名的长度大于 *cbCursorMax* 中的值，已将该游标名截断成长度为 *cbCursorMax* - 1 个字节。自变量 *pcbCursor* 包含可供返回的完整游标名的长度。函数返回 SQL_SUCCESS_WITH_INFO。

表 82. SQLGetCursorName SQLSTATE (续)

SQLSTATE	描述	说明
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *szCursor* 或 *pcbCursor* 是空指针。 对自变量 *cbCursorMax* 指定的值小于 1。
HY010	函数顺序错误	语句 *hstmt* 未处于执行状态。在调用 SQLGetCursorName() 之前，请调用 SQLExecute()、SQLExecDirect() 或 SQLSetCursorName()。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。
HY015	没有游标名可用	在 *hstmt* 上没有打开的游标，并且尚未使用 SQLSetCursorName() 来设置游标名。与 *hstmt* 相关联的语句不支持使用游标。

限制

ODBC 生成的游标名以 SQL_CUR 开头，X/Open CLI 生成的游标名以 SQLCUR 开头。DB2 UDB CLI 使用 SQLCUR。

示例

有关以下示例中使用的 check_error、initialize 和 terminate 函数的列表，请参考第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』。

```
/************************************************************************
** file = getcurs.c
**
** Example of directly executing a SELECT and positioned UPDATE SQL statement.
** Two statement handles are used, and SQLGetCursor is used to retrieve the
** generated cursor name.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLError
**        SQLExecDirect        SQLGetCursorName
************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);
```

```
int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc);

/******************************************************************
** main
** - initialize
** - terminate
******************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLRETURN   rc,
                rc2;

    rc = initialize(&henv, &hdbc);
    if (rc != SQL_SUCCESS) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt1,
                hstmt2;
     SQLCHAR    sqlstmt[]="SELECT name, job from staff for update of job";
     SQLCHAR    updstmt[MAX_STMT_LEN + 1];
     SQLCHAR    name[10],
                job[6],
                newjob[6],
                cursor[19];

     SQLINTEGER     rlength, attr;
     SQLSMALLINT    clength;

        rc = SQLAllocStmt(hdbc, &hstmt1);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        /* make sure the statement is update-capable */
        attr = SQL_FALSE;
        rc = SQLSetStmtAttr(hstmt1,SQL_ATTR_FOR_FETCH_ONLY, &attr, 0);

        /* allocate second statement handle for update statement */
        rc2 = SQLAllocStmt(hdbc, &hstmt2);
        if (rc2 != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt1, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);

        /* Get Cursor of the SELECT statement's handle */
        rc = SQLGetCursorName(hstmt1, cursor, 19, &clength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);

        /* bind name to first column in the result set */
        rc = SQLBindCol(hstmt1, 1, SQL_CHAR, (SQLPOINTER) name, 10,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);
```

```
        /* bind job to second column in the result set */
        rc = SQLBindCol(hstmt1, 2, SQL_CHAR, (SQLPOINTER) job, 6,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt1, rc);


        printf("Job Change for all clerks\n");

        while ((rc = SQLFetch(hstmt1)) == SQL_SUCCESS)
        {
            printf("Name: %-9.9s Job: %-5.5s \n", name, job);
            printf("Enter new job or return to continue\n");
            gets(newjob);
            if (newjob[0] != '\0')
            {
                sprintf( updstmt,
                    "UPDATE staff set job = '%s' where current of %s",
                    newjob, cursor);
                rc2 = SQLExecDirect(hstmt2, updstmt, SQL_NTS);
                if (rc2 != SQL_SUCCESS )
                    check_error (henv, hdbc, hstmt2, rc);
            }
        }
        if (rc != SQL_NO_DATA_FOUND )
            check_error (henv, hdbc, hstmt1, rc);
        SQLFreeStmt(hstmt1, SQL_CLOSE);
    }

    printf("Commiting Transaction\n");
    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_NO_DATA_FOUND )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
}/* end main */
```

参考

- 第 93 页的『SQLExecute — 执行语句』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 212 页的『SQLSetCursorName — 设置游标名』

SQLGetData — 从列中获取数据

用途

SQLGetData() 检索结果集的当前行中的单个列的数据。此函数是 SQLBindCol() 的备用函数，该函数导致进行 SQLFetch() 调用时将数据直接传送到应用程序变量中。SQLGetData() 还可用来以分块方式检索基于大字符的数据。

在调用 SQLGetData() 之前，必须调用 SQLFetch()。

在对每一列调用 SQLGetData() 之后，调用 SQLFetch() 来检索下一行。

SQLGetData() 与 SQLGetCol() 完全相同，支持这两个函数的目的都是为了提高兼容性。

语法

```
SQLRETURN SQLGetData (SQLHSTMT       hstmt,
                      SQLSMALLINT    icol,
                      SQLSMALLINT    fCType,
                      SQLPOINTER     rgbValue,
                      SQLINTEGER     cbValueMax,
                      SQLINTEGER     *pcbValue);
```

注: 有关适用部分的描述，请参考第 116 页的『SQLGetCol — 检索结果集的某一行的其中一列』。

SQLGetDescField — 获取描述符字段

用途

SQLGetDescField() 从描述符获取值。SQLGetDescField() 是 SQLGetDescRec() 函数的更具扩展性的备用函数。

此函数与 SQLDescribeCol() 类似，但 SQLGetDescField() 可以从参数描述符以及行描述符检索数据。

语法

```
SQLRETURN SQLGetDescField  (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLSMALLINT    fDescType,
                            SQLPOINTER     rgbDesc,
                            SQLINTEGER     bLen,
                            SQLINTEGER     *sLen);
```

函数自变量

表 83. SQLGetDescField 自变量

数据类型	自变量	用途	描述
SQLHDESC	*hdesc*	输入	描述符句柄。
SQLSMALLINT	*irec*	输入	要从中检索所指定的字段的记录的编号。
SQLSMALLINT	*fDescType*	输入	参见表 84。
SQLPOINTER	*rgbDesc*	输出	指向缓冲区的指针
SQLINTEGER	*bLen*	输入	描述符缓冲区（*rgbDesc*）的长度
SQLINTEGER *	*sLen*	输出	要返回的描述符中的实际字节数。如果此自变量包含等于或大于 *rgbDesc* 缓冲区的长度的值，则将发生截断。

表 84. fDescType 描述符类型

描述符	类型	描述
SQL_DESC_COUNT	SMALLINT	在 *rgbDesc* 中返回描述符中的记录数。
SQL_DESC_ALLOC_TYPE	SMALLINT	如果应用程序显式地分配了描述符，则为 SQL_DESC_ALLOC_USER，或者，如果实现自动地分配了描述符，则为 SQL_DESC_ALLOC_AUTO。
SQL_DESC_NAME	CHAR(128)	检索 *irec* 的 NAME 字段。
SQL_DESC_TYPE	SMALLINT	检索 *irec* 的 TYPE 字段。
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	检索类型为 SQL_DATETIME 的记录的时间间隔代码。时间间隔代码进一步地定义 SQL_DATETIME 数据类型。代码值包括 SQL_CODE_DATE、SQL_CODE_TIME 和 SQL_CODE_TIMESTAMP。

表 84. fDescType 描述符类型 (续)

描述符	类型	描述
SQL_DESC_LENGTH	INTEGER	检索 *irec* 的 LENGTH 字段。
SQL_DESC_PRECISION	SMALLINT	检索 *irec* 的 PRECISION 字段。
SQL_DESC_SCALE	SMALLINT	检索 *irec* 的 SCALE 字段。
SQL_DESC_NULLABLE	SMALLINT	如果 *irec* 可以包含空值，则在 *rgbDesc* 中返回 SQL_NULLABLE。否则，在 *rgbDesc* 中返回 SQL_NO_NULLS。
SQL_DESC_UNNAMED	SMALLINT	如果 NAME 字段是实际的名称，则此项为 SQL_NAMED，如果 NAME 字段是由实现生成的名称，则此项为 SQL_UNNAMED。
SQL_DESC_DATA_PTR	SQLPOINTER	检索 *irec* 的数据指针字段。
SQL_DESC_LENGTH_PTR	SQLPOINTER	检索 *irec* 的长度指针字段。
SQL_DESC_INDICATOR_PTR	SQLPOINTER	检索 *irec* 的指示符指针字段。

用法

如果描述符是行描述符，则描述符中的记录数与结果集中的列数相对应，对于参数描述符，与参数的数目相对应。

在 *fDescType* 设置为 SQL_DESC_COUNT 的情况下调用 `SQLGetDescField()` 是调用 `SQLNumResultCols()` 来确定是否可返回任何列的备用方法。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

诊断

表 85. SQLGetDescField SQLSTATE

SQLSTATE	描述	说明
HY009	自变量值无效	对自变量 *fDescType* 或 *irec* 指定的值无效。 自变量 *rgbDesc* 或 *sLen* 是空指针。
HY013 *****	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 186 页的『SQLPrepare — 准备语句』

SQLGetDescRec — 获取描述符记录

用途

SQLGetDescRec() 从描述符获取整个记录。SQLGetDescRec() 是 SQLDescField() 函数的更简洁的备用函数。

语法

```
SQLRETURN SQLGetDescRec    (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLCHAR        *rgbDesc,
                            SQLSMALLINT    cbDescMax,
                            SQLSMALLINT    *pcbDesc,
                            SQLSMALLINT    *type,
                            SQLSMALLINT    *subtype,
                            SQLINTEGER     *length,
                            SQLSMALLINT    *prec,
                            SQLSMALLINT    *scale,
                            SQLSMALLINT    *nullable);
```

函数自变量

表 86. SQLGetDescRec 自变量

数据类型	自变量	用途	描述
SQLHDESC	*hdesc*	输入	描述符句柄
SQLSMALLINT	*irec*	输入	要从中检索信息的记录的编号
SQLCHAR *	*rgbDesc*	输出	记录的 NAME 字段。
SQLSMALLINT	*cbDescMax*	输入	要存储在 *rgbDesc* 中的最大字节数。
SQLSMALLINT *	*pcbDesc*	输出	输出数据的总长度。
SQLSMALLINT *	*type*	输出	记录的 TYPE 字段。
SQLSMALLINT *	*subtype*	输出	对于 TYPE 为 SQL_DATETIME 的记录，此自变量为 DATETIME_INTERVAL_CODE。
SQLINTEGER *	*length*	输出	记录的 LENGTH 字段。
SQLSMALLINT *	*prec*	输出	记录的 PRECISION 字段。
SQLSMALLINT *	*scale*	输出	记录的 SCALE 字段。
SQLSMALLINT *	*nullable*	输出	记录的 NULLABLE 字段。

用法

调用 SQLGetDescRec()来检索一次调用中描述符记录中的所有数据。可能仍有必要使用 SQL_DESC_COUNT 来调用 SQLGetDescField()，以确定描述符中的记录数。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

诊断

表 87. SQLGetDescRec SQLSTATE

SQLSTATE	描述	说明
HY009	自变量值无效	对自变量 *irec* 指定的值无效。 自变量 *rgbDesc*、*pcbDesc*、*type*、*subtype*、*length*、*prec*、*scale* 或 *nullable* 是空指针。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 186 页的『SQLPrepare — 准备语句』

SQLGetDiagField — 返回诊断信息（可扩展）

用途

SQLGetDiagField() 返回与最近对特殊语句、连接或环境句柄调用的 DB2 UDB CLI 函数相关联的诊断信息。

此信息由标准化的 SQLSTATE、本机错误代码和文本消息组成。有关更多信息，请参考第 15 页的『DB2 UDB CLI 应用程序中的诊断』。

在从另一个函数调用接收到返回码 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 之后，请调用 SQLGetDiagField()。

注: 在语句的执行返回 SQL_NO_DATA_FOUND 之后，一些数据库服务器可能会提供特定于产品的诊断信息。

语法

```
SQLRETURN SQLGetDiagField (SQLSMALLINT      htype,
                           SQLINTEGER       handle,
                           SQLSMALLINT      recNum,
                           SQLSMALLINT      diagId,
                           SQLPOINTER       diagInfo,
                           SQLSMALLINT      bLen,
                           SQLSMALLINT      *sLen);
```

函数自变量

表 88. SQLDiagField 自变量

数据类型	自变量	用途	描述
SQLSMALLINT	*hType*	输入	句柄类型
SQLINTEGER	*handle*	输入	期望获取其诊断信息的句柄。
SQLSMALLINT	*recNum*	输入	如果存在多个错误，则此自变量指示应检索哪一个错误。如果请求头信息，则此自变量必须是 0。第一个错误记录编号为 1。
SQLSMALLINT	*diagId*	输入	参见表 89。
SQLPOINTER	*diagInfo*	输出	诊断信息的缓冲区。
SQLSMALLINT	*bLen*	输入	如果请求的数据是字符串，则此自变量是 *diagInfo* 的长度；否则不使用此自变量。
SQLSMALLINT *	*sLen*	输出	如果请求的数据是字符串，则此自变量是完整的诊断信息的长度；否则不使用此自变量。

表 89. diagId 类型

描述符	类型	描述
SQL_DIAG_RETURNCODE	SMALLINT	底层函数的返回码。可以是 SQL_SUCCESS、SQL_SUCCESS_WITH_INFO、SQL_NO_DATA_FOUND 或 SQL_ERROR。
SQL_DIAG_NUMBER	INTEGER	指定的句柄所带有的诊断记录的数目。

表 89. diagId 类型 (续)

描述符	类型	描述
SQL_DIAG_ROW_COUNT	INTEGER	如果句柄是语句句柄，则此变量是所指定的句柄的行数。
SQL_DIAG_SQLSTATE	CHAR(5)	与诊断记录相关的 5 个字符的 SQLSTATE 代码。SQLSTATE 代码提供可移植的诊断指示。
SQL_DIAG_NATIVE	INTEGER	与诊断记录相关的由实现定义的错误代码。可移植应用程序不应该将它们的行为基于这个值。
SQL_DIAG_MESSAGE_TEXT	CHAR(254)	与诊断记录相关的由实现定义的消息文本。
SQL_DIAG_SERVER_NAME	CHAR(128)	与诊断记录相关的服务器名，此服务器名在建立连接的 SQLConnect() 语句上提供。

用法

这些 SQLSTATE 是 X/OPEN SQL CAE 和 X/Open SQL CLI 快照定义的那些 SQLSTATE，而与特定于 IBM 的和特定于产品的 SQLSTATE 值有矛盾。

在使用同一个句柄调用除 SQLGetDiagField() 之外的函数之前，如果没有检索由一个 DB2 UDB CLI 函数生成的诊断信息，则关于前一个函数调用的信息将丢失。无论是否对第二个 DB2 UDB CLI 函数调用生成了诊断信息，情况均如此。

在给定的 DB2 UDB CLI 函数调用之后，可能有多条诊断消息可用。可通过重复地调用 SQLGetDiagField() 来检索这些消息（每次检索一条消息）。对于检索到的每条消息，SQLGetDiagField() 返回 SQL_SUCCESS 并从可用消息的列表中除去该消息。当没有更多的消息可供检索时，将返回 SQL_NO_DATA_FOUND。

当使用给定的句柄来调用 SQLGetDiagField() 时，或者当使用该句柄来进行另一个 DB2 UDB CLI 函数调用时，将清除存储在该句柄下的诊断信息。然而，使用相关联但不相同的句柄类型来调用 SQLGetDiagField() 并不会清除与给定句柄类型相关联的信息。例如，使用连接句柄输入来调用 SQLGetDiagField() 并不会清除与该连接下的任何语句句柄相关联的错误。

即使错误消息（*szDiagFieldMsg*）的缓冲区太小，也返回 SQL_SUCCESS。这是由于应用程序无法通过再次调用 SQLGetDiagField() 来检索同一错误消息。将在 *pcbDiagFieldMsg* 中返回消息文本的实际长度。

为了避免截断错误消息，请声明长度为 SQL_MAX_MESSAGE_LENGTH + 1 的缓冲区。消息文本永远不会超出这个长度。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果没有输入句柄的诊断信息可用，或者，如果已通过 SQLGetDiagField() 调用检索了所有消息，则返回 SQL_NO_DATA_FOUND。

如果自变量 diagInfo 或 sLen 是空指针，则返回 SQL_ERROR。

诊断

由于 SQLGetDiagField() 不为它自己生成诊断信息，所以没有定义 SQLSTATE。

限制

虽然 ODBC 也返回 X/Open SQL CAE SQLSTATE，但只有 DB2 UDB CLI 才返回 IBM 定义的附加 SQLSTATE。“ODBC 驱动程序管理器”还返回除标准 SQLSTATE 值之外的 SQLSTATE 值。有关特定于 ODBC 的 SQLSTATE 的更多信息，请参考 *Microsoft ODBC Programmer's Reference*。

因此，您只应该依赖于标准 SQLSTATE。这表示应用程序中的任何分支逻辑都只应该依赖于标准 SQLSTATE。有矛盾的 SQLSTATE 最适合用于调试。

SQLGetDiagRec — 返回诊断信息（简洁）

用途

SQLGetDiagRec() 返回与最近对特定语句、连接句柄和环境句柄调用的 DB2 UDB CLI 函数相关联的诊断信息。

此信息由标准化的 SQLSTATE、本机错误代码和文本消息组成。有关更多信息，请参考第 15 页的『DB2 UDB CLI 应用程序中的诊断』。

在从另一个函数调用接收到返回码 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 之后，请调用 SQLGetDiagRec()。

注: 在语句的执行返回 SQL_NO_DATA_FOUND 之后，一些数据库服务器可能会提供特定于产品的诊断信息。

语法

```
SQLRETURN SQLGetDiagRec (SQLSMALLINT  hType,
                         SQLINTEGER   handle,
                         SQLSMALLINT  recNum,
                         SQLCHAR      *szSqlState,
                         SQLINTEGER   *pfNativeError,
                         SQLCHAR      *szErrorMsg,
                         SQLSMALLINT  cbErrorMsgMax,
                         SQLSMALLINT  *pcbErrorMsg);
```

函数自变量

表 90. SQLGetDiagRec 自变量

数据类型	自变量	用途	描述
SQLSMALLINT	*hType*	输入	句柄类型
SQLINTEGER	*handle*	输入	期望获取其诊断信息的句柄。
SQLSMALLINT	*recNum*	输入	如果存在多个错误，则此自变量指示应检索哪一个错误。如果请求头信息，则此自变量必须是 0。第一个错误记录编号为 1。
SQLCHAR *	*szSqlState*	输出	SQLSTATE，作为 5 个字符的字符串返回，并且以空字符终止。前两个字符指示错误类；跟着的三个字符指示子类。这些值直接与 X/Open SQL CAE 规范和 ODBC 规范中定义的 SQLSTATE 值相对应，而与特定于 IBM 的和特定于产品的 SQLSTATE 值有矛盾。
SQLINTEGER *	*pfNativeError*	输出	本机错误代码。在 DB2 UDB CLI 中，*pfNativeError* 自变量包含 DBMS 返回的 SQLCODE 值。如果错误是由 DB2 UDB CLI 而不是由 DBMS 生成的，则将此字段设置为 -99999。
SQLCHAR *	*szErrorMsg*	输出	指向一个缓冲区的指针，该缓冲区包含由实现定义的消息文本。在 DB2 UDB CLI 中，只返回 DBMS 生成的消息；DB2 UDB CLI 本身不返回任何描述问题的消息文本。
SQLSMALLINT	*cbErrorMsgMax*	输入	缓冲区 *szErrorMsg* 的最大长度（即分配的长度）。建议的分配长度是 SQL_MAX_MESSAGE_LENGTH + 1。

表 90. SQLGetDiagRec 自变量 (续)

数据类型	自变量	用途	描述
SQLSMALLINT *	*pcbErrorMsg*	输出	指向可以返回到 *szErrorMsg* 缓冲区中的总字节数的指针。这不包括空终止字符。

用法

这些 SQLSTATE 是 X/OPEN SQL CAE 和 X/Open SQL CLI 快照定义的那些 SQLSTATE，而与特定于 IBM 的和特定于产品的 SQLSTATE 值有矛盾。

在使用同一个句柄调用除 SQLGetDiagRec() 之外的函数之前，如果没有检索由一个 DB2 UDB CLI 函数生成的诊断信息，则关于前一个函数调用的信息将丢失。无论是否对第二个 DB2 UDB CLI 函数调用生成了诊断信息，情况均如此。

在给定的 DB2 UDB CLI 函数调用之后，可能有多条诊断消息可用。可通过重复地调用 SQLGetDiagRec() 来检索这些消息（每次检索一条消息）。对于检索到的每条消息，SQLGetDiagRec() 返回 SQL_SUCCESS 并从可用消息的列表中除去该消息。当没有更多的消息可供检索时，将返回 SQL_NO_DATA_FOUND，并将 SQLSTATE 设置为 “00000”，将 *pfNativeError* 设置为 0，而 *pcbErrorMsg* 和 *szErrorMsg* 未定义。

当使用给定的句柄来调用 SQLGetDiagRec() 时，或者当使用该句柄来进行另一个 DB2 UDB CLI 函数调用时，将清除存储在该句柄下的诊断信息。然而，使用相关联但不相同的句柄类型来调用 SQLGetDiagRec() 并不会清除与给定句柄类型相关联的信息。例如，使用连接句柄输入来调用 SQLGetDiagRec() 并不会清除与该连接下的任何语句句柄相关联的错误。

即使错误消息的缓冲区（*szErrorMsg*）太短也会返回 SQL_SUCCESS，这是因为应用程序无法通过再次调用 SQLGetDiagRec() 来检索同一错误消息。将在 *pcbErrorMsg* 中返回消息文本的实际长度。

为了避免截断错误消息，请声明长度为 SQL_MAX_MESSAGE_LENGTH + 1 的缓冲区。消息文本永远不会超出这个长度。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果没有输入句柄的诊断信息可用，或者，如果已通过 SQLGetDiagRec() 调用检索了所有消息，则返回 SQL_NO_DATA_FOUND。

如果自变量 szSqlState、pfNativeError、szErrorMsg 或 pcbErrorMsg 是空指针，则返回 SQL_ERROR。

诊断

由于 SQLGetDiagRec() 不为它自己生成诊断信息，所以没有定义 SQLSTATE。

限制

虽然 ODBC 也返回 X/Open SQL CAE SQLSTATE，但只有 DB2 UDB CLI 才返回 IBM 定义的附加 SQLSTATE。“ODBC 驱动程序管理器”还返回除标准 SQLSTATE 值之外的 SQLSTATE 值。有关特定于 ODBC 的 SQLSTATE 的更多信息，请参考 *Microsoft ODBC Programmer's Reference*。

因此，您只应该依赖于标准 SQLSTATE。这表示应用程序中的任何分支逻辑都只应该依赖于标准 SQLSTATE。有矛盾的 SQLSTATE 最适合用于调试。

参考

• 第 133 页的『SQLGetDiagField — 返回诊断信息（可扩展）』

SQLGetEnvAttr — 返回环境属性的当前设置

用途

SQLGetEnvAttr() 返回所指定的环境属性的当前设置。

这些选项是使用 SQLSetEnvAttr() 函数设置的。

语法

```
SQLRETURN SQLGetEnvAttr (SQLHENV      henv,
                         SQLINTEGER   Attribute,
                         SQLPOINTER   Value,
                         SQLINTEGER   BufferLength,
                         SQLINTEGER   *StringLength);
```

函数自变量

表 91. SQLGetEnvAttr 自变量

数据类型	自变量	用途	描述
SQLHENV	*henv*	输入	环境句柄
SQLINTEGER	*Attribute*	输入	要检索的属性。有关更多信息，请参考第 218 页的表 159。
SQLPOINTER	*Value*	输出	与 *Attribute* 相关联的当前值。所返回的值的类型要视 *Attribute* 而定。
SQLINTEGER	*BufferLength*	输入	如果属性值是字符串，则自变量是 *Value* 指向的缓冲区的最大大小；否则不使用此自变量。
SQLINTEGER *	*StringLength*	输出	如果属性值是字符串，则此自变量是输出数据的长度（字节）；否则不使用此自变量。

如果 *Attribute* 表示的不是字符串，则 DB2 UDB CLI 忽略 *BufferLength*，并且不设置 *StringLength*。

用法

在分配环境句柄与释放环境句柄之间的任何时候都可以调用 SQLGetEnvAttr()。此函数用于获取环境属性的当前值。

诊断

表 92. SQLGetEnvAttr SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	属性超出范围	指定了无效的 *Attribute* 值。 自变量 *Value* 或 *StringLength* 是空指针。

SQLGetFunctions — 获取函数

用途

SQLGetFunctions() 用于查询是否支持特定的函数。这使应用程序在使用不同的驱动程序时能够适应不断变化的支持级别。

在调用此函数之前，必须调用 SQLConnect()，并且必须存在与数据库（数据库服务器）的连接。

语法

```
SQLRETURN SQLGetFunctions (SQLHDBC        hdbc,
                           SQLSMALLINT    fFunction,
                           SQLSMALLINT    *pfSupported);
```

函数自变量

表 93. SQLGetFunctions 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	数据库连接句柄
SQLSMALLINT	*fFunction*	输入	正在查询的函数
SQLSMALLINT *	*pfSupported*	输出	指向一个位置的指针，根据是否支持所查询的函数，此函数在该位置返回 SQL_TRUE 或 SQL_FALSE。

用法

图 4 显示了 *fFunction* 自变量的有效值以及是否支持相对应的函数。

注: 当连接到远程服务器时，不支持标有星号的值。

```
SQL_API_ALLOCCONNECT      = TRUE
SQL_API_ALLOCENV          = TRUE
SQL_API_ALLOCHANDLE       = TRUE
SQL_API_ALLOCSTMT         = TRUE
SQL_API_BINDCOL           = TRUE
SQL_API_BINDFILETOCOL     = TRUE
SQL_API_BINDFILETOPARAM   = TRUE
SQL_API_BINDPARAM         = TRUE
SQL_API_BINDPARAMETER     = TRUE
SQL_API_CANCEL            = TRUE
SQL_API_CLOSECURSOR       = TRUE
```

图 4. 支持的函数 (1/2)

```
SQL_API_COLATTRIBUTES     = TRUE
SQL_API_COLUMNS           = TRUE
SQL_API_CONNECT           = TRUE
SQL_API_COPYDESC          = TRUE
SQL_API_DATASOURCES       = TRUE
SQL_API_DESCRIBECOL       = TRUE
SQL_API_DESCRIBEPARAM     = TRUE
SQL_API_DISCONNECT        = TRUE
SQL_API_DRIVERCONNECT     = TRUE
SQL_API_ENDTRAN           = TRUE
SQL_API_ERROR             = TRUE
SQL_API_EXECDIRECT        = TRUE
SQL_API_EXECUTE           = TRUE
SQL_API_EXTENDEDFETCH     = TRUE
SQL_API_FETCH             = TRUE
SQL_API_FOREIGNKEYS       = TRUE
SQL_API_FREECONNECT       = TRUE
SQL_API_FREEENV           = TRUE
SQL_API_FREEHANDLE        = TRUE
SQL_API_FREESTMT          = TRUE
SQL_API_GETCOL            = TRUE
SQL_API_GETCONNECTATTR    = TRUE
SQL_API_GETCONNECTOPTION  = TRUE
SQL_API_GETCURSORNAME     = TRUE
SQL_API_GETDATA           = TRUE
SQL_API_GETDESCFIELD      = TRUE
SQL_API_GETDESCREC        = TRUE
SQL_API_GETDIAGFIELD      = TRUE
SQL_API_GETDIAGREC        = TRUE
SQL_API_GETENVATTR        = TRUE
SQL_API_GETFUNCTIONS      = TRUE
SQL_API_GETINFO           = TRUE
SQL_API_GETLENGTH         = TRUE
SQL_API_GETPOSITION       = TRUE
SQL_API_GETSTMTATTR       = TRUE
SQL_API_GETSTMTOPTION     = TRUE
SQL_API_GETSUBSTRING      = TRUE
SQL_API_GETTYPEINFO       = TRUE
SQL_API_LANGUAGES         = TRUE
SQL_API_MORERESULTS       = TRUE
SQL_API_NATIVESQL         = TRUE
SQL_API_NUMPARAMS         = TRUE
SQL_API_NUMRESULTCOLS     = TRUE
SQL_API_PARAMDATA         = TRUE
SQL_API_PARAMOPTIONS      = TRUE
SQL_API_PREPARE           = TRUE
SQL_API_PRIMARYKEYS       = TRUE
SQL_API_PROCEDURECOLUMNS  = TRUE
SQL_API_PROCEDURES        = TRUE
SQL_API_PUTDATA           = TRUE
SQL_API_RELEASEENV        = TRUE
SQL_API_ROWCOUNT          = TRUE
SQL_API_SETCONNECTATTR    = TRUE
SQL_API_SETCONNECTOPTION  = TRUE
SQL_API_SETCURSORNAME     = TRUE
SQL_API_SETDESCFIELD      = TRUE
SQL_API_SETDESCREC        = TRUE
SQL_API_SETENVATTR        = TRUE
SQL_API_SETPARAM          = TRUE
SQL_API_SETSTMTATTR       = TRUE
SQL_API_SETSTMTOPTION     = TRUE
SQL_API_SPECIALCOLUMNS    = TRUE *
SQL_API_STATISTICS        = TRUE *
SQL_API_TABLES            = TRUE
SQL_API_TRANSACT          = TRUE
```

图 4. 支持的函数 (2/2)

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 94. SQLGetFunctions SQLSTATE

SQLSTATE	描述	说明
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *pfSupported* 是空指针。
HY010	函数顺序错误。肯定是尚未分配连接句柄。	在 SQLConnect 之前调用了 SQLGetFunctions。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

SQLGetInfo — 获取一般信息

用途

SQLGetInfo() 返回关于应用程序当前所连接的 DBMS 的一般信息（包括支持的数据转换）。

语法

```
SQLRETURN SQLGetInfo (SQLHDBC        hdbc,
                      SQLSMALLINT    fInfoType,
                      SQLPOINTER     rgbInfoValue,
                      SQLSMALLINT    cbInfoValueMax,
                      SQLSMALLINT    *pcbInfoValue);
```

函数自变量

表 95. SQLGetInfo 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	数据库连接句柄
SQLSMALLINT	*fInfoType*	输入	所期望的信息的类型
SQLPOINTER	*rgbInfoValue*	输出（也是输入）	指向一个缓冲区的指针，此函数将在该缓冲区中存储期望的信息。根据所检索的信息的类型的不同，可返回 4 个字节的信息: • 16 位整数值 • 32 位整数值 • 32 位二进制值 • 以空终止的字符串
SQLSMALLINT	*cbInfoValueMax*	输入	*rgbInfoValue* 指针指向的缓冲区的最大长度。
SQLSMALLINT *	*pcbInfoValue*	输出	指向一个位置的指针，此函数将在该位置中返回可返回的期望信息的总字节数。 如果 *pcbInfoValue* 指向的位置中的值大于 *cbInfoValueMax* 中指定的 *rgbInfoValue* 缓冲区大小，则将把字符串输出信息截断成长度为 *cbInfoValueMax* - 1 字节，并且此函数将返回 SQL_SUCCESS_WITH_INFO。

用法

表 96 列示了 *fInfoType* 的可能值以及 SQLGetInfo() 对该值将返回的信息的描述。

表 96. SQLGetInfo 返回的信息

fInfoType	格式	描述及注意事项
SQL_ACTIVE_CONNECTIONS	短整数	每个应用程序支持的最大活动连接数。 返回零，指示限制依赖于系统资源。
SQL_ACTIVE_STATEMENTS	短整数	每个连接的最大活动语句数。 返回零，指示限制依赖于系统资源。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_AGGREGATE_FUNCTIONS	32 位掩码	一个位掩码，它枚举对聚集函数的支持: • SQL_AF_ALL • SQL_AF_AVG • SQL_AF_COUNT • SQL_AF_DISTINCT • SQL_AF_MAX • SQL_AF_MIN • SQL_AF_SUM
SQL_CATALOG_NAME	字符串	字符串“Y”指示服务器支持目录名。“N”指示不支持目录名。
SQL_COLUMN_ALIAS	字符串	连接是否支持列别名。如果连接支持列别名这一概念，则返回值“Y”。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_CONVERT_BIGINT SQL_CONVERT_BINARY SQL_CONVERT_BLOB SQL_CONVERT_CHAR SQL_CONVERT_CLOB SQL_CONVERT_DATE SQL_CONVERT_DBCLOB SQL_CONVERT_DECIMAL SQL_CONVERT_DOUBLE SQL_CONVERT_FLOAT SQL_CONVERT_INTEGER SQL_CONVERT_LONGVARBINARY SQL_CONVERT_LONGVARCHAR SQL_CONVERT_NUMERIC SQL_CONVERT_REAL SQL_CONVERT_SMALLINT SQL_CONVERT_TIME SQL_CONVERT_TIMESTAMP SQL_CONVERT_VARBINARY SQL_CONVERT_VARCHAR SQL_CONVERT_WCHAR SQL_CONVERT_WLONGVARCHAR SQL_CONVERT_WVARCHAR	32 位掩码	指示使用 CONVERT 标量函数时数据源对具有 infoType 中命名的类型的数据所支持的转换。如果此掩码等于零，则表示数据源对具有所命名的类型的数据不支持任何转换，这包括面向同一数据类型的转换。 例如，如果要了解数据源是否支持 SQL_INTEGER 数据到 SQL_DECIMAL 数据类型的转换，应用程序应调用 SQLGetInfo()，并将 finfoType 设置为 SQL_CONVERT_INTEGER。然后，应用程序对返回的掩码和 SQL_CVT_DECIMAL 进行 AND 运算。如果得到的值非零，则表示支持该转换。使用下列掩码来确定支持哪些转换: • SQL_CVT_BIGINT • SQL_CVT_BINARY • SQL_CONVERT_BLOB • SQL_CVT_CHAR • SQL_CONVERT_CLOB • SQL_CVT_DATE • SQL_CONVERT_DBCLOB • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_FLOAT • SQL_CVT_INTEGER • SQL_CVT_LONGVARBINARY • SQL_CVT_LONGVARCHAR • SQL_CVT_NUMERIC • SQL_CVT_REAL • SQL_CONVERT_SMALLINT • SQL_CONVERT_TIME • SQL_CONVERT_TIMESTAMP • SQL_CONVERT_VARBINARY • SQL_CONVERT_VARCHAR • SQL_CONVERT_WCHAR • SQL_CONVERT_WLONGVARCHAR • SQL_CONVERT_WVARCHAR
SQL_CONVERT_FUNCTIONS	32 位掩码	指示驱动程序支持的标量转换函数和相关联的数据源: • SQL_FN_CVT_CONVERT — 用来确定支持哪些转换函数。 • SQL_FN_CVT_CAST — 用来确定支持哪些强制类型转换函数。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_CORRELATION_NAME	短整数	指示服务器支持相关名的程度。 • SQL_CN_ANY — 支持，并且可以是任何有效的用户定义名称。 • SQL_CN_NONE — 不支持相关名。 • SQL_CN_DIFFERENT — 支持相关名，但它必须与它表示的表的名称不同。
SQL_CURSOR_COMMIT_BEHAVIOR	16 位整数	指示 COMMIT 操作如何影响游标。值及其含义列示如下: • SQL_CB_DELETE — 破坏游标并删除动态 SQL 语句的访问计划。 • SQL_CB_CLOSE — 破坏游标，但保留动态 SQL 语句（包括非查询语句）的访问计划 • SQL_CB_PRESERVE — 保留游标和动态语句（包括非查询语句）的访问计划。应用程序可以继续取装数据，也可以关闭游标并重新执行查询（无需重新准备语句）。 注意: 在 COMMIT 之后 — 在可以执行诸如定位型更新或删除之类的操作之前，必须发出 FETCH 以重新定位游标。
SQL_CURSOR_ROLLBACK_BEHAVIOR	16 位整数	指示 ROLLBACK 操作如何影响游标。值及其含义列示如下: • SQL_CB_DELETE — 破坏游标并删除动态 SQL 语句的访问计划。 • SQL_CB_CLOSE — 破坏游标，但保留动态 SQL 语句（包括非查询语句）的访问计划 • SQL_CB_PRESERVE — 保留游标和动态语句（包括非查询语句）的访问计划。应用程序可以继续取装数据，也可以关闭游标并重新执行查询（无需重新准备语句）。 注意: DB2 服务器没有 SQL_CB_PRESERVE 特性。
SQL_DATA_SOURCE_NAME	字符串	连接句柄的已连接的数据源的名称。
SQL_DATA_SOURCE_READ_ONLY	字符串	字符串 “Y” 指示已将数据库设置为 READ ONLY 方式; “N” 指示未将其设置为 READ ONLY 方式。
SQL_DBMS_NAME	字符串	正在访问的 DBMS 产品的名称。 例如: • QSQ 表示 “DB2 UDB iSeries 版” • SQL 表示 “DB2 UDB OS/2 版” • DSN 表示 “DB2 UDB zOS 和 OS/390 版”
SQL_DBMS_VER	字符串	所访问的 DBMS 的版本。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_DEFAULT_TXN_ISOLATION	32 位掩码	所支持的缺省事务隔离级别。 返回下列其中一个掩码: • SQL_TXN_READ_UNCOMMITTED — 所有事务都立即接收到更改（脏读取、不可重复读取和幻象读取都是有可能的）。 这等价于 UR 级别。 • SQL_TXN_READ_COMMITTED — 事务 2 可以改变和落实事务 1 读取的行（不可重复读取和幻象读取是有可能的）。 这等价于 CS 级别。 • SQL_TXN_REPEATABLE_READ — 事务可以添加或除去与搜索条件或暂挂事务相匹配的行（可重复读取，但幻象读取是有可能的）。 这等价于 RS 级别。 • SQL_TXN_SERIALIZABLE — 受暂挂事务影响的数据不可供其它事务使用（可重复读取和幻象读取是不可能的）。 这等价于 RR 级别。 • SQL_TXN_VERSIONING — 不适用于 IBM DBMS。 • SQL_TXN_NOCOMMIT — 在成功的操作结束时有效地落实任何更改；不允许显式地落实或回滚。 这是 DB2 UDB iSeries 版隔离级别。 在 IBM 术语中， • SQL_TXN_READ_UNCOMMITTED 是指未落实的读取; • SQL_TXN_READ_COMMITTED 是指游标稳定性; • SQL_TXN_REPEATABLE_READ 是指读取稳定性; • SQL_TXN_SERIALIZABLE 是指可重复读取。
SQL_DESCRIBE_PARAMETER	字符串	如果可以描述参数，则此项为 Y；否则为 N。
SQL_DRIVER_NAME	字符串	用来访问数据源的驱动程序的文件名。
SQL_DRIVER_ODBC_VER	字符串	驱动程序所支持的 ODBC 的版本号。DB2 ODBC 返回 `2.1`。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_GROUP_BY	16 位整数	指示服务器对 GROUP BY 子句的支持程度: • SQL_GB_NO_RELATION — GROUP BY 中的列与 SELECT 列表中的列之间不存在关系。 • SQL_GB_NOT_SUPPORTED — 不支持 GROUP BY。 • SQL_GB_GROUP_BY_EQUALS_SELECT — GROUP BY 必须包括 SELECT 列表中的所有非聚集列。 • SQL_GB_GROUP_BY_CONTAINS_SELECT — GROUP BY 子句必须包含 SELECT 列表中的所有非聚集列。
SQL_IDENTIFIER_CASE	16 位整数	指示对象名（如表名）的区分大小写。 值及其含义列示如下: • SQL_IC_UPPER — 标识符名称以大写形式存储在系统目录中。 • SQL_IC_LOWER — 标识符名称以小写形式存储在系统目录中。 • SQL_IC_SENSITIVE — 标识符名称区分大小写，并且以混合大小写形式存储在系统目录中。 • SQL_IC_MIXED — 标识符名称不区分大小写，并且以混合大小写形式存储在系统目录中。 注意: IBM DBMS 中的标识符名称不区分大小写。
SQL_IDENTIFIER_QUOTE_CHAR	字符串	用作用引号括起来的字符串的定界符的字符。
SQL_LIKE_ESCAPE_CLAUSE	字符串	一个字符串，这个字符串指示是否支持将转义字符用作 LIKE 谓词中的元字符百分号和下划线。
SQL_MAX_CATALOG_NAME_LEN	16 位整数	目录限定符名的最大长度; 由 3 部分组成的表名的第一部分（字节）。
SQL_MAX_COLUMN_NAME_LEN	短整数	列名的最大长度。
SQL_MAX_COLUMNS_IN_GROUP_BY	短整数	GROUP BY 子句中的最大列数。
SQL_MAX_COLUMNS_IN_INDEX	短整数	SQL 索引中的最大列数。
SQL_MAX_COLUMNS_IN_ORDER_BY	短整数	ORDER BY 子句中的最大列数。
SQL_MAX_COLUMNS_IN_SELECT	短整数	SELECT 语句中的最大列数。
SQL_MAX_COLUMNS_IN_TABLE	短整数	SQL 表中的最大列数。
SQL_MAX_CURSOR_NAME_LEN	短整数	游标名的最大长度。
SQL_MAX_OWNER_NAME_LEN	短整数	所有者名称的最大长度。
SQL_MAX_ROW_SIZE	32 位无符号整数	指定服务器在基本表的单一行中支持的最大长度（字节）。如果没有限制，则为零。
SQL_MAX_SCHEMA_NAME_LEN	整数	模式名的最大长度。
SQL_MAX_STATEMENT_LEN	32 位无符号整数	指示 SQL 语句字符串的最大长度（字节），这包括语句中的空格数。
SQL_MAX_TABLE_NAME	短整数	表名的最大长度。
SQL_MAX_TABLES_IN_SELECT	短整数	SELECT 语句中的最大表数。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_MULTIPLE_ACTIVE_TXN	字符串	字符串 “Y” 指示允许在多个连接上有活动事务。 “N” 指示每次只有一个连接可以有活动事务。
SQL_NON_NULLABLE_COLUMNS	16 位整数	指示是否支持非空列: • SQL_NNC_NON_NULL — 可将列定义为非空。 • SQL_NNC_NULL — 不能将列定义为非空。
SQL_NUMERIC_FUNCTIONS	32 位掩码	指示所支持的标量数字函数。 使用下列掩码来确定支持哪些数字函数: • SQL_FN_NUM_ABS • SQL_FN_NUM_ACOS • SQL_FN_NUM_ASIN • SQL_FN_NUM_ATAN • SQL_FN_NUM_ATAN2 • SQL_FN_NUM_CEILING • SQL_FN_NUM_COS • SQL_FN_NUM_COT • SQL_FN_NUM_DEGREES • SQL_FN_NUM_EXP • SQL_FN_NUM_FLOOR • SQL_FN_NUM_LOG • SQL_FN_NUM_LOG10 • SQL_FN_NUM_MOD • SQL_FN_NUM_PI • SQL_FN_NUM_POWER • SQL_FN_NUM_RADIANS • SQL_FN_NUM_RAND • SQL_FN_NUM_ROUND • SQL_FN_NUM_SIGN • SQL_FN_NUM_SIN • SQL_FN_NUM_SQRT • SQL_FN_NUM_TAN • SQL_FN_NUM_TRUNCATE
SQL_ODBC_API_CONFORMANCE	16 位整数	ODBC 一致性级别: • SQL_OAC_NONE • SQL_OAC_LEVEL1 • SQL_OAC_LEVEL2
SQL_ODBC_SQL_CONFORMANCE	16 位整数	值及其含义列示如下: • SQL_OSC_MINIMUM — 表示支持最少量的 ODBC SQL 语法 • SQL_OSC_CORE — 表示支持核心 ODBC SQL 语法 • SQL_OSC_EXTENDED — 表示支持扩展 ODBC SQL 语法 要了解上述三种类型的 ODBC SQL 语法的定义，请阅读 Microsoft ODBC 3.0 Software Development Kit and Programmer’s Reference。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_ORDER_BY_COLUMNS_IN_SELECT	字符串	如果 ORDER BY 子句中的列必须包含在 SEELCT 列表中，则将此项设置为 “Y”；否则设置为 “N”。
SQL_OUTER_JOINS	字符串	字符串及其含义如下: • “Y” 指示支持外连接，并且 DB2 ODBC 支持 ODBC 外连接请求语法。 • “N” 指示不支持外连接。
SQL_OWNER_TERM 或 SQL_SCHEMA_TERM	字符串	模式（所有者）的数据库提供方术语。
SQL_OWNER_USAGE 或 SQL_SCHEMA_USAGE	32 位掩码	指示在执行这些 SQL 语句时具有相关联的模式（所有者）的 SQL 语句的类型。模式限定符（所有者）列示如下: • SQL_OU_DML_STATEMENTS — 在所有 DML 语句中都受支持。 • SQL_OU_PROCEDURE_INVOCATION — 在过程调用语句中受支持。 • SQL_OU_TABLE_DEFINITION — 在所有表定义语句中都受支持。 • SQL_OU_INDEX_DEFINITION — 在所有索引定义语句中都受支持。 • SQL_OU_PRIVILEGE_DEFINITION — 在所有特权定义语句（即授权和取消语句）中都受支持。
SQL_POSITIONED_STATEMENTS	32 位掩码	指示对定位型 UPDATE 和定位型 DELETE 语句的支持程度: • SQL_PS_POSITIONED_DELETE • SQL_PS_POSITIONED_UPDATE • SQL_PS_SELECT_FOR_UPDATE，指示服务器是否要求在 <查询表达式> 中指定 FOR UPDATE 子句才允许通过游标更新列。
SQL_PROCEDURE_TERM	字符串	过程的数据源名。
SQL_PROCEDURES	字符串	当前服务器是否支持 SQL 过程。如果连接支持 SQL 过程，则返回 “Y” 值。
SQL_QUALIFIER_LOCATION 或 SQL_CATALOG_LOCATION	16 位整数	16 位整数值，这个值指示限定符在限定表名中的位置。零指示不支持限定名。
SQL_QUALIFIER_NAME_SEPARATOR 或 SQL_CATALOG_NAME_SEPARATOR	字符串	用作目录名与其后的限定名元素之间的分隔符的字符。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_QUALIFIER_TERM 或 SQL_CATALOG_TERM	字符串	限定符的数据库提供方术语。 提供方对三部分名称的高位部分使用的名称。 由于 DB2 ODBC 不支持三部分的名称，所以返回长度为零的字符串。 对于非 ODBC 应用程序，应该使用 SQL_CATALOG_TERM 符号名来代替 SQL_QUALIFIER_NAME。
SQL_QUALIFIER_USAGE 或 SQL_CATALOG_USAGE	32 位掩码	除了用于目录之外，此项与 SQL_OWNER_USAGE 类似。
SQL_QUOTED_IDENTIFIER_CASE	16 位整数	返回: • SQL_IC_UPPER — SQL 中用引号括起来的标识符不区分大小写，并且以大写形式存储在系统目录中。 • SQL_IC_LOWER — SQL 中用引号括起来的标识符不区分大小写，并且以小写形式存储在系统目录中。 • SQL_IC_SENSITIVE — SQL 中用引号括起来的标识符（定界标识符）区分大小写，并且以混合大小写形式存储在系统目录中。 • SQL_IC_MIXED — SQL 中用引号括起来的标识符不区分大小写，并且以混合大小写形式存储在系统目录中。 您应该将此项与用来确定如何在系统目录中存储（用引号括起来的）标识符的 SQL_IDENTIFIER_CASE fInfoType 作对比。
SQL_SEARCH_PATTERN_ESCAPE	字符串	用来指定驱动程序支持将哪些字符用作编目函数（如 SQLTables() 和 SQLColumns()）的转义字符。
SQL_SQL92_PREDICATES	32 位掩码	指示 SQL-92 定义的在 SELECT 语句中受支持的谓词。 • SQL_SP_BETWEEN • SQL_SP_COMPARISON • SQL_SP_EXISTS • SQL_SP_IN • SQL_SP_ISNOTNULL • SQL_SP_ISNULL • SQL_SP_LIKE • SQL_SP_MATCH_FULL • SQL_SP_MATCH_PARTIAL • SQL_SP_MATCH_UNIQUE_FULL • SQL_SP_MATCH_UNIQUE_PARTIAL • SQL_SP_OVERLAPS • SQL_SP_QUANTIFIED_COMPARISON • SQL_SP_UNIQUE

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_SQL92_VALUE_EXPRESSIONS	32 位掩码	指示由 SQL-92 定义的受支持的值表达式。 • SQL_SVE_CASE • SQL_SVE_CAST • SQL_SVE_COALESCE • SQL_SVE_NULLIF
SQL_STRING_FUNCTIONS	32 位掩码	指示支持哪些字符串函数。 使用下列位掩码来确定支持哪些字符串函数: • SQL_FN_STR_ASCII • SQL_FN_STR_CHAR • SQL_FN_STR_CONCAT • SQL_FN_STR_DIFFERENCE • SQL_FN_STR_INSERT • SQL_FN_STR_LCASE • SQL_FN_STR_LEFT • SQL_FN_STR_LENGTH • SQL_FN_STR_LOCATE • SQL_FN_STR_LOCATE_2 • SQL_FN_STR_LTRIM • SQL_FN_STR_REPEAT • SQL_FN_STR_REPLACE • SQL_FN_STR_RIGHT • SQL_FN_STR_RTRIM • SQL_FN_STR_SOUNDEX • SQL_FN_STR_SPACE • SQL_FN_STR_SUBSTRING • SQL_FN_STR_UCASE 如果应用程序可以使用 string1、string2 和 start 自变量来调用 LOCATE 标量函数，则将返回 SQL_FN_STR_LOCATE 位掩码。如果应用程序只可以使用 string1 和 string2 来调用 LOCATE 标量函数，则返回 SQL_FN_STR_LOCATE_2 位掩码。如果完全支持 LOCATE 标量函数，则同时返回这两个位掩码。

表 96. SQLGetInfo 返回的信息 (续)

fInfoType	格式	描述及注意事项
SQL_TIMEDATE_FUNCTIONS	32 位掩码	指示支持哪些时间和日期函数。 使用下列位掩码来确定支持哪些日期函数: • SQL_FN_TD_CURDATE • SQL_FN_TD_CURTIME • SQL_FN_TD_DAYNAME • SQL_FN_TD_DAYOFMONTH • SQL_FN_TD_DAYOFWEEK • SQL_FN_TD_DAYOFYEAR • SQL_FN_TD_HOUR • SQL_FN_TD_JULIAN_DAY • SQL_FN_TD_MINUTE • SQL_FN_TD_MONTH • SQL_FN_TD_MONTHNAME • SQL_FN_TD_NOW • SQL_FN_TD_QUARTER • SQL_FN_TD_SECOND • SQL_FN_TD_SECONDS_SINCE_MIDNIGHT • SQL_FN_TD_TIMESTAMPADD • SQL_FN_TD_TIMESTAMPDIFF • SQL_FN_TD_WEEK • SQL_FN_TD_YEAR
SQL_TXN_CAPABLE	短整数	指示事务是否可以包含 DDL 和／或 DML: • SQL_TC_NONE — 不支持事务。 • SQL_TC_DML — 事务只可以包含 DML 语句（SELECT、INSERT、UPDATE 和 DELETE, 等等）。在事务中遇到的 DDL 语句（CREATE TABLE 和 DROP INDEX, 等等）将导致错误。 • SQL_TC_DDL_COMMIT — 事务只可以包含 DML 语句。在事务中遇到的 DDL 语句将导致落实事务。 • SQL_TC_DDL_IGNORE — 事务只可以包含 DML 语句。将忽略在事务中遇到的 DDL 语句。 • SQL_TC_ALL — 事务可以包含任何次序的 DDL 和 DML 语句。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLGetInfo

诊断

表 97. SQLGetInfo SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	请求的信息是作为以空终止的字符串返回的，它的长度超出 *cbInfoValueMax* 中指定的应用程序缓冲区长度。自变量 *pcbInfoValue* 包含所请求的信息的实际（未截断）长度。
08003	连接未打开	在 *fInfoType* 中请求的信息的类型需要已打开的连接。只有 SQL_ODBC_VER 不需要已打开的连接。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *rgbInfoValue* 是空指针 指定了无效的 *fInfoType*。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

SQLGetLength — 检索字符串值的长度

用途

SQLGetLength() 用来检索当前事务期间（作为取装或 SQLGetSubString() 调用的结果）从服务器返回的大对象定位器所引用的大对象值的长度。

语法

```
SQLRETURN  SQLGetLength      (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       LocatorCType,
                              SQLINTEGER        Locator,
                              SQLINTEGER        *StringLength,
                              SQLINTEGER        *IndicatorValue);
```

函数自变量

表 98. SQLGetLength 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。这可以是任何已分配的但当前尚未对其指定已准备的语句的语句句柄。
SQLSMALLINT	*LocatorCType*	输入	源 LOB 定位器的 C 类型。这可以是: • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	输入	必须设置为 LOB 定位器值。
SQLINTEGER *	*StringLength*	输出	指定的定位器的长度。[a] 如果将指针设置为空，则将返回 SQLSTATE **HY**009。
SQLINTEGER *	*IndicatorValue*	输出	总是设置为零。

注: a. 这是以字节计的，即使对于 DBCLOB 数据也如此。

用法

SQLGetLength() 可用来确定 LOB 定位器所表示的数据值的长度。应用程序使用此函数来确定所引用的 LOB 值的总长度，以便可以选择适当的策略来获取某些或全部 LOB 值。

Locator 自变量可以包含任何既没有使用 FREE LOCATOR 语句显式释放也没有因为创建定位器的事务已终止而隐式释放的有效定位器。

语句句柄一定不能已经与任何已准备的语句或编目函数调用相关联。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 99. SQLGetLength SQLSTATE

SQLSTATE	描述	说明
07006	转换无效	*LocatorCType* 与 *Locator* 的组合无效。
58004	意外的系统故障	不可恢复的系统错误。
HY003	程序类型超出范围	*LocatorCType* 不是 SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR 或 SQL_C_DBCLOB_LOCATOR 的其中之一。
HY009	自变量值无效	自变量 *StringLength* 或 *IndicatorValue* 是空指针。
HY010	函数顺序错误	指定的 *StatementHandle* 未处于*已分配*状态。
HYC00	驱动程序不具有能力	应用程序当前连接到不支持大对象的数据源。
0F001	LOB 变量无效	对 *Locator* 指定的值尚未与 LOB 定位器相关联。

限制

当连接到不支持“大对象”的 DB2 服务器时，此函数不可用。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 157 页的『SQLGetPosition — 返回字符串的开始位置』
- 第 163 页的『SQLGetSubString — 检索字符串值的一部分』

SQLGetPosition — 返回字符串的开始位置

用途

SQLGetPosition() 用来返回一个字符串在 LOB 值（源）中的开始位置。源值必须是 LOB 定位器，搜索字符串可以是 LOB 定位器或文字字符串。

源与搜索 LOB 定位器可以是取装操作或 SQLGetSubString() 调用在当前事务期间从数据库返回的任何 LOB 定位器。

语法

```
SQLRETURN   SQLGetPosition   (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       LocatorCType,
                              SQLINTEGER        SourceLocator,
                              SQLINTEGER        SearchLocator,
                              SQLCHAR           *SearchLiteral,
                              SQLINTEGER        SearchLiteralLength,
                              SQLINTEGER        FromPosition,
                              SQLINTEGER        *LocatedAt,
                              SQLINTEGER        *IndicatorValue);
```

函数自变量

表 100. SQLGetPosition 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。这可以是任何已分配的但当前尚未对其指定已准备的语句的语句句柄。
SQLSMALLINT	*LocatorCType*	输入	源 LOB 定位器的 C 类型。这可以是: • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*SourceLocator*	输入	必须将 *SourceLocator* 设置为源 LOB 定位器。
SQLINTEGER	*SearchLocator*	输入	如果 *SearchLiteral* 指针是空，并且如果 *SearchLiteralLength* 设置为 0，则必须将 *SearchLocator* 设置为与搜索字符串相关联的 LOB 定位器；否则，将忽略此字符串。
SQLCHAR *	*SearchLiteral*	输入	此自变量指向包含搜索字符串文字的存储区域。 如果 *SearchLiteralLength* 为 0，则此指针必须为空。
SQLINTEGER	*SearchLiteralLength*	输入	*SearchLiteral* 中的字符串的长度（字节）。[a] 如果此自变量值为 0，则自变量 *SearchLocator* 有意义。
SQLINTEGER	*FromPosition*	输入	对于 BLOB 和 CLOB，这是源字符串中开始进行搜索的第一个字节的位置。对于 DBCLOB，这是第一个字节。开始字节或字符编号为 1。

表 100. SQLGetPosition 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER *	*LocatedAt*	输出	对于 BLOB 和 CLOB，这是找到字符串的字节位置；或者找不到，则值为零。对于 DBCLOB，这是字符位置。 如果源字符串的长度是零，则返回 1 值。
SQLINTEGER *	*IndicatorValue*	输出	始终设置为零。

注:

a 这是以字节计的，即使对于 DBCLOB 数据也如此。

用法

将 SQLGetPosition() 与 SQLGetSubString() 配合使用可以以随机方式获取字符串的任意部分。为了使用 SQLGetSubString()，必须事先了解子串在整个字符串中的位置。在搜索字符串能够找到该子串的开始位置的情况下，可使用 SQLGetPosition() 来获取该子串的开始位置。

Locator 和 *SearchLocator*（如果使用的话）自变量可以包含任何既没有使用 FREE LOCATOR 语句显式释放也没有因为创建定位器的事务已终止而隐式释放的有效定位器。

Locator 和 *SearchLocator* 必须具有相同的 LOB 定位器类型。

语句句柄一定不能已经与任何已准备的语句或编目函数调用相关联。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 101. SQLGetPosition SQLSTATE

SQLSTATE	描述	说明
07006	转换无效	*LocatorCType* 与任一个 LOB 定位器值的组合无效。
42818	长度无效	模式的长度太长。
58004	意外的系统故障	不可恢复的系统错误。
HY009	自变量值无效	自变量 *LocatedAt* 或 *IndicatorValue* 是空指针。 *FromPosition* 的自变量值不大于 0。 *LocatorCType* 不是 SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR 或 SQL_C_DBCLOB_LOCATOR 的其中之一。
HY010	函数顺序错误	指定的 *StatementHandle* 未处于*已分配*状态。
HY090	字符串或缓冲区长度无效	*SearchLiteralLength* 的值小于 1，并且不是 SQL_NTS。

表 101. SQLGetPosition SQLSTATE (续)

SQLSTATE	描述	说明
HYC00	驱动程序不具有能力	应用程序当前连接到不支持大对象的数据源。
0F001	LOB 变量无效	对 *Locator* 或 *SearchLocator* 指定的值当前不是 LOB 定位器。

限制

当连接到不支持“大对象”的 DB2 服务器时，此函数不可用。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 95 页的『SQLExtendedFetch — 取装行数组』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 155 页的『SQLGetLength — 检索字符串值的长度』
- 第 163 页的『SQLGetSubString — 检索字符串值的一部分』

SQLGetStmtAttr — 获取语句属性的值

用途

SQLGetStmtAttr() 返回所指定的语句属性的当前设置。

这些选项是使用 SQLSetStmtAttr() 函数设置的。此函数与 SQLGetStmtOption() 类似，支持这两个函数的目的都是为了提高兼容性。

语法

```
SQLRETURN SQLGetStmtAttr( SQLHSTMT        hstmt,
                          SQLINTEGER      fAttr,
                          SQLPOINTER      pvParam,
                          SQLINTEGER      bLen,
                          SQLINTEGER      *sLen);
```

函数自变量

表 102. SQLGetStmtAttr 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLINTEGER	*fAttr*	输入	要检索的属性。有关更多信息，请参考表 103。
SQLPOINTER	*pvParam*	输出	指向所请求的属性的缓冲区的指针。
SQLINTEGER	*bLen*	输入	如果属性是字符串，则此自变量是要在 *pvParam* 中存储的最大字节数；否则不使用此变量。
SQLINTEGER *	*sLen*	输出	如果属性是字符串，则此自变量是输出数据的长度；否则不使用此变量。

用法

表 103. 语句属性

fAttr	数据类型	内容
SQL_ATTR_FOR_FETCH_ONLY	整数	指示为此语句句柄打开的游标是否应该是只读的。 • SQL_FALSE — 游标可用于定位型更新和删除。这是缺省值。 • SQL_TRUE — 游标是只读的，并且不能用于定位型更新或删除。
SQL_ATTR_APP_ROW_DESC	整数	描述符句柄，应用程序通过此描述符句柄来使用语句句柄检索行数据。
SQL_ATTR_APP_PARAM_DESC	整数	描述符句柄，应用程序使用此描述符句柄来为此语句句柄提供参数值。

表 103. 语句属性 (续)

fAttr	数据类型	内容
SQL_ATTR_CURSOR_SCROLLABLE	整数	32 位的整数值，这个值指定为此语句句柄打开的游标是否应该可滚动。 • SQL_FALSE — 游标不可滚动，不能再次对它们使用 SQLFetchScroll()。这是缺省值。 • SQL_TRUE — 游标可滚动。可使用 SQLFetchScroll() 来从这些游标检索数据。
SQL_ATTR_CURSOR_TYPE	整数	32 位的整数值，这个值指定为此语句句柄打开的游标的行为。 • SQL_CURSOR_FORWARD_ONLY — 游标不可滚动，不能再次对它们使用 SQLFetchScroll()。这是缺省值。 • SQL_DYNAMIC — 游标可滚动。可使用 SQLFetchScroll() 来从这些游标检索数据。
SQL_ATTR_IMP_ROW_DESC	整数	描述符句柄，CLI 实现使用此描述符句柄来通过使用此语句句柄检索行数据。
SQL_ATTR_IMP_PARAM_DESC	整数	描述符句柄，CLI 实现使用此描述符句柄来为此语句句柄提供参数值。
SQL_ATTR_ROWSET_SIZE	整数	32 位的整数值，这个值指定行集中的行数。这是每个 SQLExtendedFetch() 调用所返回的行数。缺省值是 1。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 104. SQLStmtOption SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *pvParam* 是空指针。 指定了无效的 *fAttr* 值。
HYC00	驱动程序不具有能力	DB2 UDB CLI 能够识别此选项，但不支持它。

SQLGetStmtOption — 返回语句选项的当前设置

用途

SQLGetStmtOption() 返回所指定的语句选项的当前设置。

这些选项是使用 SQLSetStmtOption() 函数设置的。

语法

```
SQLRETURN SQLGetStmtOption( SQLHSTMT        hstmt,
                            SQLSMALLINT     fOption,
                            SQLPOINTER      pvParam);
```

函数自变量

表 105. SQLStmtOption 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	连接句柄
SQLSMALLINT	*fOption*	输入	要检索的选项。有关更多信息，请参考第 160 页的表 103。
SQLPOINTER	*pvParam*	输出	选项的值。根据 *fOption* 的值的不同，这可以是 32 位整数值，也可以是指向以空终止的字符串的指针。

用法

SQLGetStmtOption() 与 SQLGetStmtAttr() 提供相同的功能，支持这两个函数的目的都是为了提高兼容性。

有关语句选项的列表，参见第 160 页的表 103。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 106. SQLStmtOption SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *pvParam* 是空指针。 指定了无效的 *fOption* 值。
HYC00	驱动程序不具有能力	DB2 UDB CLI 能够识别此选项，但不支持它。

SQLGetSubString — 检索字符串值的一部分

用途

SQLGetSubString() 用来检索当前事务期间（由取装操作或前一个 SQLGetSubString() 调用）从服务器返回的大对象定位器所引用的大对象值的一部分。

语法

```
SQLRETURN  SQLGetSubString   (
                SQLHSTMT          StatementHandle,
                SQLSMALLINT       LocatorCType,
                SQLINTEGER        SourceLocator,
                SQLINTEGER        FromPosition,
                SQLINTEGER        ForLength,
                SQLSMALLINT       TargetCType,
                SQLPOINTER        DataPtr,
                SQLINTEGER        BufferLength,
                SQLINTEGER        *StringLength,
                SQLINTEGER        *IndicatorValue);
```

函数自变量

表 107. SQLGetSubString 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。这可以是任何已分配的但当前尚未对其指定已准备的语句的语句句柄。
SQLSMALLINT	*LocatorCType*	输入	源 LOB 定位器的 C 类型。这可以是: • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*SourceLocator*	输入	必须将 *SourceLocator* 设置为源 LOB 定位器值。
SQLINTEGER	*FromPosition*	输入	对于 BLOB 和 CLOB，这是函数要返回的第一个字节的位置。对于 DBCLOB，这是第一个字节。开始字节或字符编号为 1。
SQLINTEGER	*ForLength*	输入	这是函数将要返回的字符串的长度。对于 BLOB 和 CLOB，这是以字节计的长度。对于 DBCLOB，这是以字符计的长度。 如果 *FromPosition* 小于源字符串长度，但 *FromPosition* + *ForLength* - 1 超出源字符串的末尾，则在结果的右端填充必需数目个字符（对于 BLOB，填充 X'00'，对于 CLOB，填充单字节空白字符，对于 DBCLOB，填充双字节空白字符）。
SQLSMALLINT	*TargetCType*	输入	*DataPtr* 的 C 数据类型。目标必须是 C 字符串变量（SQL_C_CHAR、SQL_C_WCHAR、SQL_C_BINARY 或 SQL_C_DBCHAR）。
SQLPOINTER	*DataPtr*	输出	指向一个缓冲区的指针，将在该缓冲区中存储检索到的字符串值或 LOB 定位器。
SQLINTEGER	*BufferLength*	输入	*DataPtr* 指向的缓冲区的最大大小（字节）。

表 107. SQLGetSubString 自变量 (续)

数据类型	自变量	用途	描述
SQLINTEGER *	*StringLength*	输出	如果目标 C 缓冲区类型打算用于二进制或字符串变量，而非用于定位器值，则此自变量是 *DataPtr* 中返回的信息的长度（字节）[a]。 如果此指针设置为空，则不返回任何内容。
SQLINTEGER *	*IndicatorValue*	输出	始终设置为零。

注:

a 这是以字节计的，即使对于 DBCLOB 数据也如此。

用法

SQLGetSubString() 用来获取 LOB 定位器所表示的字符串的任何一部分。可以选择两个目标:

- 目标可以是适当的 C 字符串变量。
- 可以在服务器上创建新的 LOB 值，可以将该值的 LOB 定位器赋给客户机上的目标应用程序变量。

可以将 SQLGetSubString() 用作 SQLGetData 的备用函数来以分块方式获取数据。在此情况下，首先将列绑定到 LOB 定位器，然后使用 LOB 定位器来取装整个 LOB 或以分块方式取装 LOB。

Locator 自变量可以包含任何既没有使用 FREE LOCATOR 语句显式释放也没有因为创建定位器的事务已终止而隐式释放的有效定位器。

语句句柄一定不能已经与任何已准备的语句或编目函数调用相关联。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 108. SQLGetSubString SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	要返回的数据的长度大于 *BufferLength*。可供返回的实际长度存储在 *StringLength* 中。
07006	转换无效	对 *TargetCType* 指定的值不是 SQL_C_CHAR、SQL_C_BINARY、SQL_C_DBCHAR 或 LOB 定位器。 对 *TargetCType* 指定的值不适合于源（例如，SQL_C_DBCHAR 不适合于 BLOB 列）。
22011	发生子串错误	*FromPosition* 大于源字符串的长度。
58004	意外的系统故障	不可恢复的系统错误。

表 108. SQLGetSubString SQLSTATE (续)

SQLSTATE	描述	说明
HY003	程序类型超出范围	*LocatorCType* 不是 SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR 或 SQL_C_DBCLOB_LOCATOR 的其中之一。
HY009	自变量值无效	对 *FromPosition* 或 *ForLength* 指定的值不是正整数。 自变量 *DataPtr*、*StringLength* 或 *IndicatorValue* 是空指针。
HY010	函数顺序错误	指定的 *StatementHandle* 未处于*已分配*状态。
HY090	字符串或缓冲区长度无效	*BufferLength* 的值小于 0。
HYC00	驱动程序不具有能力	应用程序当前连接到不支持大对象的数据源。
0F001	当前未指定定位器	对 *Locator* 指定的值当前不是 LOB 定位器。

限制

当连接到不支持“大对象”的 DB2 服务器时，此函数不可用。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 97 页的『SQLFetch — 取装下一行』
- 第 127 页的『SQLGetData — 从列中获取数据』
- 第 155 页的『SQLGetLength — 检索字符串值的长度』
- 第 157 页的『SQLGetPosition — 返回字符串的开始位置』

SQLGetTypeInfo — 获取数据类型信息

用途

SQLGetTypeInfo() 返回关于与 DB2 UDB CLI 相关联的 DBMS 所支持的数据类型的信息。将在 SQL 结果集中返回此信息。可使用用来处理查询的那些函数来接收列。

语法

```
SQLRETURN    SQLGetTypeInfo    (SQLHSTMT          StatementHandle,
                                SQLSMALLINT       DataType);
```

函数自变量

表 109. SQLGetTypeInfo 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLSMALLINT	*DataType*	输入	所查询的 SQL 数据类型。支持的类型包括: • SQL_ALL_TYPES • SQL_BIGINT • SQL_CHAR • SQL_DATE • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TIME • SQL_TIMESTAMP • SQL_VARCHAR • SQL_VARGRAPHIC 如果指定 SQL_ALL_TYPES，则将按 TYPE_NAME 的升序返回关于所有受支持的数据类型的信息。结果集不会包含所有不受支持的数据类型。

用法

由于 SQLGetTypeInfo() 将生成一个结果集，并且等同于执行查询，所以它将生成游标并开始事务。要在此语句句柄上准备并执行另一个语句，必须关闭游标。

如果使用无效的 *DataType* 来调用 SQLGetTypeInfo()，则将返回空结果集。

下面描述了此函数生成的结果集的各个列。

虽然在将来的发行版中可能会添加新列，并可能会更改现有列的名称，但当前列的位置不会更改。所返回的数据类型就是可以在 CREATE TABLE、ALTER TABLE 和 DDL 语句中使用的那些数据类型。不持久的数据类型不是所返回的结果集的一部分。也不返回用户定义的数据类型。

表 110. SQLGetTypeInfo 返回的列

列号 / 列名	数据类型	描述
1 TYPE_NAME	VARCHAR(128) 非空	SQL 数据类型名的字符表示法（例如，VARCHAR、DATE 和 INTEGER）。
2 DATA_TYPE	SMALLINT 非空	SQL 数据类型定义值（例如，SQL_VARCHAR、SQL_DATE 和 SQL_INTEGER）。
3 COLUMN_SIZE	INTEGER	如果数据类型是字符或二进制字符串，则此列包含最大长度（字节）；如果是图形（DBCS）字符串，则这是列的双字节字符数。 对于日期、时间和时间戳记数据类型，这是转换为字符后显示值所必需的总字符数。 对于数字数据类型，这是总位数。
4 LITERAL_PREFIX	VARCHAR(128)	一个字符，DB2 将该字符识别为具有此数据类型的文字的前缀。对于文字前缀不适用的数据类型，此列为空。
5 LITERAL_SUFFIX	VARCHAR(128)	一个字符，DB2 将该字符识别为具有此数据类型的文字的后缀。对于文字后缀不适用的数据类型，此列为空。
6 CREATE_PARAMS	VARCHAR(128)	此列的文本包含由逗号分隔的关键字列表，这些关键字与应用程序在 TYPE_NAME 列中使用名称作为 SQL 中的数据类型时可以在括号内指定的每个参数相对应。列表中的关键字可以是下列任何一个关键字: LENGTH、PRECISION 或 SCALE。这些关键字按照 SQL 语法所要求的使用次序出现。 如果数据类型定义没有参数（如 INTEGER），则返回空指示符。 **注**: CREATE_PARAMS 的意图是使应用程序能够定制 *DDL 构建器*的接口。使用 CREATE_PARAMS，应用程序应只能确定定义数据类型所必需的自变量的数目，以及得到用于标注编辑控件的本地化文本。
7 NULLABLE	SMALLINT 非空	指示数据类型是否接受空值 • 如果不允许空值，则设置为 SQL_NO_NULLS。 • 如果允许空值，则设置为 SQL_NULLABLE。
8 CASE_SENSITIVE	SMALLINT 非空	指示是否可将数据类型视为区分大小写以便进行整理；有效值是 SQL_TRUE 和 SQL_FALSE。

表 110. SQLGetTypeInfo 返回的列 (续)

列号 / 列名	数据类型	描述
9 SEARCHABLE	SMALLINT 非空	指示如何在 WHERE 子句中使用该数据类型。有效值包括: • SQL_UNSEARCHABLE — 如果不能在 WHERE 子句中使用该数据类型的话。 • SQL_LIKE_ONLY — 如果只能在 WHERE 子句中将该数据类型与 LIKE 谓词配合使用的话。 • SQL_ALL_EXCEPT_LIKE — 如果可以在 WHERE 子句中将该数据类型与除 LIKE 之外的所有比较运算符配合使用的话。 • SQL_SEARCHABLE — 如果可以在 WHERE 子句中将该数据类型与任何比较运算符配合使用的话。
10 UNSIGNED_ATTRIBUTE	SMALLINT	指示数据类型是否是无符号的。有效值是: SQL_TRUE、SQL_FALSE 或 NULL。如果此属性不适用于数据类型，则返回空指示符。
11 FIXED_PREC_SCALE	SMALLINT 非空	如果数据类型是精确数字数据类型并且始终具有相同的精度和标度，则此列包含 SQL_TRUE 值；否则，此列包含 SQL_FALSE。
12 AUTO_INCREMENT	SMALLINT	如果在插入行时将把具有此数据类型的列自动设置为唯一的值，则此列包含 SQL_TRUE; 否则此列包含 SQL_FALSE。
13 LOCAL_TYPE_NAME	VARCHAR(128)	此列包含数据类型的任何与该数据类型的规则名不同的本地化（本机语言）名称。如果没有本地化名称，则此列为空。 此列仅用于显示。字符串的字符集依赖于语言环境，并且通常是数据库的缺省字符集。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 111. SQLGetTypeInfo SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	已在语句句柄上打开了游标。尚未关闭 *StatementHandle*。
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY004	SQL 数据类型超出范围	指定了无效的 *DataType*。
HY010	函数顺序错误	在执行数据（`SQLParamData()` 和 `SQLPutData()`）操作中调用了此函数。
HYT00	超时到期	

限制

任何 IBM RDBMS 都不支持下列由 ODBC 指定的 SQL 数据类型（以及它们的对应 *DataType* 定义值）：

数据类型	*DataType*
TINY INT	SQL_TINYINT
BIT	SQL_BIT

示例

```
/* From CLI sample typeinfo.c */
/* ... */
    rc = SQLGetTypeInfo(hstmt, SQL_ALL_TYPES);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, (SQLPOINTER) typename.s, 128, &typename.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_DEFAULT, (SQLPOINTER) & datatype,
                    sizeof(datatype), &datatype_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_DEFAULT, (SQLPOINTER) & precision,
                    sizeof(precision), &precision_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_DEFAULT, (SQLPOINTER) & nullable,
                    sizeof(nullable), &nullable_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 8, SQL_C_DEFAULT, (SQLPOINTER) & casesens,
                    sizeof(casesens), &casesens_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    printf("Datatype                  Datatype Precision  Nullable   Case\n");
    printf("Typename                   (int)                      Sensitive\n");
    printf("------------------------- -------- ---------- -------- ---------\n");
    /* LONG VARCHAR FOR BIT DATA      99  2147483647   FALSE    FALSE */
    /* Fetch each row, and display */
    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
        printf("%-25s ", typename.s);
        printf("%8d ", datatype);
        printf("%10ld ", precision);
        printf("%-8s ", truefalse[nullable]);
        printf("%-9s\n", truefalse[casesens]);
    }                           /* endwhile */

    if ( rc != SQL_NO_DATA_FOUND )
       CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 143 页的『SQLGetInfo — 获取一般信息』

SQLLanguages — 获取 SQL 方言或一致性信息

用途

SQLLanguages() 用于返回 SQL 方言或一致性信息。将在 SQL 结果集中返回此信息，可以使用那些用来取装由 SELECT 语句生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLLanguages  (SQLHSTMT       hstmt);
```

函数自变量

表 112. SQLLanguages 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄

用法

此函数使用 StatementHandle 指定的结果集来返回方言和一致性信息。对于 SQL 产品所作的每项一致性声称（包括为 ISO 和特定于提供方的版本定义的子集），此结果集都包含一行。因此，对于声称符合此规范的产品，结果集包含至少一行。

用于定义 ISO 标准和特定于提供方的语言的行可存在于同一个表中。每一行都至少带有这些列，并且，如果进行 X/Open SQL 一致性声称，则各个列包含这些值。

表 113. SQLLanguages 返回的列

列名	数据类型	描述
SOURCE	VARCHAR(254)，非空	定义此 SQL 版本的组织。
SOURCE_YEAR	VARCHAR(254)	批准相关源文档的年份。
CONFORMANCE	VARCHAR(254)	实现所声称的与相关文档一致的级别。
INTEGRITY	VARCHAR(254)	对实现是否支持“完整性增强功能”（IEF）的指示。
IMPLEMENTATION	VARCHAR(254)	提供方定义的字符串，这个字符串唯一地标识提供方的 SQL 产品。
BINDING_SYTLE	VARCHAR(254)	“EMBEDDED”、“DIRECT”或“CLI”。
PROGRAMMING_LANG	VARCHAR(254)	对其支持绑定样式的主语言。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 114. SQLLanguages SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	请求了与游标相关的信息，但没有打开游标。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HYC00	驱动程序不具有能力	DB2 UDB CLI 不支持将 *catalog* 用作表名的限定符。

SQLMoreResults — 确定是否有更多的结果集

用途

SQLMoreResults() 确定是否有更多的有关已经与正在返回结果集的存储过程相关联的语句句柄的信息可用。

语法

```
SQLRETURN   SQLMoreResults   (SQLHSTMT          StatementHandle);
```

函数自变量

表 115. SQLMoreResults 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。

用法

此函数用来返回多个在执行包含 SQL 查询的存储过程之后按顺序设置的结果。在存储过程完成执行之后，已使游标保持打开状态，以便可以继续访问结果集。

在彻底地处理第一个结果集之后，应用程序可以调用 SQLMoreResults() 来确定是否有另一个结果集可用。如果当前结果集有未取装的行，则 SQLMoreResults() 通过关闭游标来将它们废弃，并且，如果有另一个结果集可用，则返回 SQL_SUCCESS。

如果所有结果集都已处理完毕，则 SQLMoreResults() 返回 SQL_NO_DATA_FOUND。

如果使用 SQL_CLOSE 或 SQL_DROP 选项来调用 SQLFreeStmt()，则将废弃此语句句柄上的所有暂挂结果集。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

错误状态

表 116. SQLMoreResults SQLSTATE

SQLSTATE	描述	说明
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
58004	意外的系统故障	不可恢复的系统错误。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。
HY013	意外的内存处理错误	DB2 UDB CLI 无法访问支持此函数的执行或完成所必需的内存。

表 116. SQLMoreResults SQLSTATE (续)

SQLSTATE	描述	说明
HYT00	超时到期	

此外，SQLMoreResults() 可以返回与 SQLExecute() 相关联的 SQLSTATE。

限制

SQLMoreResults() 的 ODBC 规范还允许返回与带有输入参数值数组的参数化的 INSERT、UPDATE 和 DELETE 语句的执行相关联的计数。然而，DB2 UDB CLI 不支持返回这样的计数信息。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 49 页的『SQLBindParameter — 将参数标记绑定到缓冲区』

SQLNativeSql — 获取本机 SQL 文本

用途

SQLNativeSql() 用来显示 DB2 UDB CLI 如何解释提供方转义子句。如果应用程序传入的原始 SQL 字符串包含提供方转义子句序列，则 DB2 UDB CLI 返回数据源将会看到的经过转换的 SQL 字符串（适当地对提供方转义子句进行转换或将其废弃）。

语法

```
SQLRETURN   SQLNativeSql     (SQLHDBC           ConnectionHandle,
                              SQLCHAR           *InStatementText,
                              SQLINTEGER        TextLength1,
                              SQLCHAR           *OutStatementText,
                              SQLINTEGER        BufferLength,
                              SQLINTEGER        *TextLength2Ptr);
```

函数自变量

表 117. SQLNativeSql 自变量

数据类型	自变量	用途	描述
SQLHDBC	ConnectionHandle	输入	连接句柄
SQLCHAR *	InStatementText	输入	输入 SQL 字符串
SQLINTEGER	TextLength1	输入	*InStatementText* 的长度
SQLCHAR *	OutStatementText	输出	指向经过转换的输出字符串的缓冲区的指针
SQLINTEGER	BufferLength	输入	*OutStatementText* 所指向的缓冲区的大小
SQLINTEGER *	TextLength2Ptr	输出	可以在 *OutStatementText* 中返回的总字节数。如果可供返回的字节数大于或等于 *BufferLength*，则将 *OutStatementText* 中的输出 SQL 字符串截断成长度为 *BufferLength - 1* 字节。如果没有生成输出字符串，则将返回 SQL_NULL_DATA 值。

用法

当应用程序希望检查或显示 DB2 UDB CLI 将要传送给数据源的经过转换的 SQL 字符串时，应该调用此函数。仅当输入 SQL 语句字符串包含提供方转义子句序列时才会发生转换（映射）。

在 iSeries 上不存在提供方转义序列；提供此过程的目的只是为了提高兼容性。并且请注意，此过程可用来评估 SQL 字符串是否含有语法错误。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 118. SQLNativeSql SQLSTATE

SQLSTATE	描述	说明
01004	已截断数据	缓冲区 *OutStatementText* 不够大，无法包含整个 SQL 字符串，因此发生了截断。自变量 *TextLength2Ptr* 包含未截断的 SQL 字符串的总长度。（函数将返回 SQL_SUCCESS_WITH_INFO）
08003	连接已关闭	*ConnectionHandle* 未引用打开的数据库连接。
37000	SQL 语法无效	*InStatementText* 中的输入 SQL 字符串包含语法错误。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *InStatementText*、*OutStatementText* 或 *TextLength2Ptr* 是空指针。
HY090	字符串或缓冲区长度无效	自变量 *TextLength1* 小于 0，但不等于 SQL_NTS。 自变量 *BufferLength* 小于 0。

限制

无。

示例

```
/* From CLI sample native.c */
/* ... */
    SQLCHAR in_stmt[1024], out_stmt[1024] ;
    SQLSMALLINT pcPar ;
    SQLINTEGER indicator ;
/* ... */
    /* Prompt for a statement to prepare */
    printf("Enter an SQL statement: \n");
    gets((char *)in_stmt);

    /* prepare the statement */
    rc = SQLPrepare(hstmt, in_stmt, SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    SQLNumParams(hstmt, &pcPar);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    SQLNativeSql(hstmt, in_stmt, SQL_NTS, out_stmt, 1024, &indicator);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    if ( indicator == SQL_NULL_DATA ) printf( "Invalid statement\n" ) ;
    else {
       printf( "Input Statement: \n %s \n", in_stmt ) ;
       printf( "Output Statement: \n %s \n", in_stmt ) ;
       printf( "Number of Parameter Markers = %d\n", pcPar ) ;
    }

    rc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;
```

参考

无。

SQLNextResult — 处理下一个结果集

用途

SQLNextResult() 确定是否有更多的有关已经与正在返回结果集的存储过程相关联的语句句柄的信息可用。

语法

```
SQLRETURN   SQLNextResult   (SQLHSTMT          StatementHandle,
                             SQLHSTMT          NextResultHandle);
```

函数自变量

表 119. SQLNextResult 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLHSTMT	NextResultHandle	输入	下一个结果集的语句句柄。

用法

此函数用来使来自 StatementHandle 的下一个结果集与 NextResultHandle 相关联。这与 SQLMoreResults() 有所不同，其原因在于此函数允许两个语句句柄同时处理它们的结果集。

如果所有结果集都已处理完毕，则 SQLNextResult() 返回 SQL_NO_DATA_FOUND。

如果使用 SQL_CLOSE 或 SQL_DROP 选项来调用 SQLFreeStmt()，则将废弃此语句句柄上的所有暂挂结果集。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

错误状态

表 120. SQLNextResult SQLSTATE

SQLSTATE	描述	说明
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
58004	意外的系统故障	不可恢复的系统错误。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。
HY013	意外的内存处理错误	DB2 UDB CLI 无法访问支持此函数的执行或完成所必需的内存。
HYT00	超时到期	

参考

- 第 172 页的『SQLMoreResults — 确定是否有更多的结果集』

SQLNumParams — 获取 SQL 语句中的参数数目

用途

SQLNumParams() 返回 SQL 语句中的参数标记数目。

语法

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       *ParameterCountPtr);
```

函数自变量

表 121. SQLNumParams 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLSMALLINT *	ParameterCountPtr	输出	语句中的参数数目。

用法

只有在准备了与 *StatementHandle* 相关联的语句之后才可以调用此函数。如果该语句不包含任何参数标记，则将把 *ParameterCountPtr* 设置为 0。

应用程序可以调用此函数来确定需要为与该语句句柄相关联的 SQL 语句调用多少次 SQLBindParameter()。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 122. SQLNumParams SQLSTATE

SQLSTATE	描述	说明
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY008	已将操作取消	
HY009	自变量值无效	*ParameterCountPtr* 为空。
HY010	函数顺序错误	在对指定的 *StatementHandle* 调用 SQLPrepare() 之前调用了此函数 在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。
HY013	意外的内存处理错误	DB2 UDB CLI 无法访问支持此函数的执行或完成所必需的内存。
HYT00	超时到期	

限制

无。

示例

请参考 SQLNativeSql()（第 175 页的『示例』）。

参考

- 第 44 页的『SQLBindParam — 将缓冲区绑定到参数标记』
- 第 186 页的『SQLPrepare — 准备语句』

SQLNumResultCols — 获取结果列数

用途

SQLNumResultCols() 返回与输入语句句柄相关联的结果集中的列数。

在调用此函数之前，必须调用 SQLPrepare() 或 SQLExecDirect()。

在调用此函数之后，可以调用 SQLDescribeCol()、SQLColAttributes()、SQLBindCol() 或 SQLGetData()。

语法

```
SQLRETURN SQLNumResultCols (SQLHSTMT       hstmt,
                            SQLSMALLINT    *pccol);
```

函数自变量

表 123. SQLNumResultCols 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT *	*pccol*	输出	结果集中的列数

用法

如果对输入语句句柄执行的上一个语句不是 SELECT，则此函数将把输出自变量设置为零。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 124. SQLNumResultCols SQLSTATE

SQLSTATE	描述	说明
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	*pcbCol* 是空指针。
HY010	函数顺序错误	在对 *hstmt* 调用 SQLPrepare 或 SQLExecDirect 之前调用了此函数。
S1013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 58 页的『SQLColAttributes — 列属性』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 116 页的『SQLGetCol — 检索结果集的某一行的其中一列』
- 第 186 页的『SQLPrepare — 准备语句』

SQLParamData — 获取下一个需要其数据值的参数

用途

SQLParamData() 与 SQLPutData() 一起用来以分块方式发送长数据。也可使用此函数来发送定长数据。

语法

```
SQLRETURN SQLParamData  (SQLHSTMT     hstmt,
                         SQLPOINTER   *prgbValue);
```

函数自变量

表 125. SQLParamData 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLPOINTER *	*prgbValue*	输出	指向 SQLSetParam 调用上指定的 *rgbValue* 自变量的值的指针。

用法

如果存在至少一个还没有为其指定数据的 SQL_DATA_AT_EXEC 参数，则 SQLParamData() 返回 SQL_NEED_DATA。此函数在应用程序在上一次 SQLBindParam() 调用期间提供的 *prgbValue* 中返回应用程序定义的值。将 SQLPutData() 调用一次或多次以发送参数数据。然后，调用 SQLParamData() 来指示已经为当前参数发送了所有数据，并且前进到下一个 SQL_DATA_AT_EXEC 参数。在对所有参数指定数据值并且成功地执行相关联的语句之后，将返回 SQL_SUCCESS。如果在实际的语句执行期间或之前发生任何错误，则返回 SQL_ERROR。

如果 SQLParamData() 返回 SQL_NEED_DATA，则只能进行 SQLPutData() 或 SQLCancel() 调用。所有其它使用此语句句柄的函数调用都将失败。另外，对于引用 *hstmt* 的父 *hdbc* 的函数调用，如果它们涉及更改该连接的任何属性或状态，则它们全都会失败。对父 *hdbc* 进行的下列函数调用也是不允许的:

- SQLAllocConnect()
- SQLAllocHandle()
- SQLAllocStmt()
- SQLSetConnectOption()

万一在 SQL_NEED_DATA 序列中调用它们，这些函数将返回 SQL_ERROR，并且 SQLSTATE 为 **HY**010，SQL_DATA_AT_EXEC 参数的处理不受影响。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA

诊断

SQLParamData() 可以返回 SQLExecDirect() 和 SQLExecute() 函数所返回的任何 SQLSTATE。另外，还可以生成下列诊断:

表 126. SQLParamData SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *prgbValue* 是空指针。
HY010	函数顺序错误	调用 SQLParamData() 的顺序不正确。此调用只有在 SQLExecDirect() 或 SQLExecute() 之后或者在 SQLPutData() 调用之后才是有效的。
HYDE0	没有数据处于执行值暂挂状态	尽管在 SQLExecDirect() 或 SQLExecute() 调用之后调用了此函数，但却没有（剩下）SQL_DATA_AT_EXEC 参数可供处理。

SQLParamOptions — 指定参数的输入数组

用途

SQLParamOptions() 提供了为 SQLBindParameter() 设置的每个参数设置多个值的能力。这使应用程序能够通过调用一次 SQLExecute() 或 SQLExecDirect() 来将多行插入到表中。

语法

```
SQLRETURN   SQLParamOptions  (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Crow,
                              SQLINTEGER        *FetchOffsetPtr);
```

函数自变量

表 127. SQLParamOptions 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLINTEGER	*Crow*	输入	每个参数的值数目。如果此自变量大于 1，则 SQLBindParameter() 中的 *rgbValue* 自变量指向参数值数组，并且 *pcbValue* 指向长度数组。
SQLINTEGER *	*FetchOffsetPtr*	输出（延迟）	当前不使用此自变量。

用法

此函数可以与 SQLBindParameter() 一起用来设置多行 INSERT 语句。为了完成此操作，应用程序必须为所有正在插入的数据分配存储器。必须以行方向的方式组织此数据。这表示第一行的所有数据都是连续的，后面跟着下一行的所有数据，等等。应使用 SQLBindParameter() 函数来绑定所有输入参数类型和长度。对于多行 INSERT 语句的情况，将使用 SQLBindParameter() 上提供的地址来引用第一行数据。通过不断地使地址递增整一行的长度，将可以引用所有的后续行。

例如，应用程序打算将 100 行的数据插入到表中，每行包含 4 个字节的整数值，后跟 10 个字节的字符值。应用程序将分配 1400 个字节的存储器，并使用适当的行数据来填充每个 14 字节的存储块。

并且，SQLBindParameter() 上传送的指示符指针必须引用 800 个字节的存储块。此存储块用来传入任何空指示符值。此存储器也是行方向的，因此前 8 个字节是两个用于第一行的指示符，后跟两个用于下一行的指示符，依此类推。应用程序使用 SQLParamOptions() 函数来指定下次执行使用该语句句柄的 INSERT 语句时将要插入的行数。INSERT 语句必须具有多行格式。

例如: INSERT INTO CORPDATA.NAMES ? ROWS VALUES(?, ?)

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 128. SQLParamOptions SQLSTATE

SQLSTATE	描述	说明
HY009	自变量值无效	自变量 *Crow* 中的值小于 1。
HY010	函数顺序错误	在执行数据（SQLParamData() 和 SQLPutData()）操作中调用了此函数。

限制

无。

参考

- 第 44 页的『SQLBindParam — 将缓冲区绑定到参数标记』
- 第 172 页的『SQLMoreResults — 确定是否有更多的结果集』

SQLPrepare — 准备语句

用途

SQLPrepare() 使 SQL 语句与输入语句句柄相关联并将该语句发送至 DBMS 以进行准备。应用程序可以通过将该语句句柄传送给其它函数来引用这个准备好的语句。

如果已将语句句柄与 SELECT 语句配合使用，则在调用 SQLPrepare() 之前，必须调用 SQLFreeStmt() 来关闭游标。

语法

```
SQLRETURN SQLPrepare (SQLHSTMT       hstmt,
                      SQLCHAR        *szSqlStr,
                      SQLINTEGER     cbSqlStr);
```

函数自变量

表 129. SQLPrepare 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄。一定不能有打开的游标与 *hstmt* 相关联。
SQLCHAR *	*szSqlStr*	输入	SQL 语句字符串
SQLINTEGER	*cbSqlStr*	输入	*szSqlStr* 自变量的内容的长度。 必须将此长度设置为 *szSqlstr* 中的 SQL 语句的精确长度，或者，如果语句文本以空终止，则设置为 SQL_NTS。

用法

在使用 SQLPrepare() 准备语句之后，应用程序可通过调用下列函数来请求关于结果集的格式的信息（如果该语句是 SELECT 语句的话）：

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttributes()

可通过调用 SQLExecute() 来将准备好的语句执行一次或多次。在将该语句句柄与另一个 SQLPrepare()、SQLExecDirect()、SQLColumns()、SQLSpecialColumns()、SQLStatistics() 或 SQLTables() 配合使用之前，该 SQL 语句将保持与该句柄相关联。

SQL 语句字符串可包含参数标记。参数标记由 “?” 字符表示，它用来指示语句中的一个位置，在调用 SQLExecute() 时，将在该位置替代应用程序变量的值。SQLBindParam() 用来将应用程序变量绑定到每个参数标记（使应用程序变量与参数标记相关联），以及指示在传送数据时是否应该执行任何数据转换。

SQL 语句不能是 COMMIT 或 ROLLBACK。必须通过调用 SQLTransact() 来发出 COMMIT 或 ROLLBACK。

如果 SQL 语句是定位型 DELETE 或定位型 UPDATE，则必须在同一个连接句柄下的单独语句句柄上定义该语句所引用的游标。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 130. SQLPrepare SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	在指定的 *hstmt* 上存在打开的游标。
37xxx	语法错误或访问违例	*szSqlStr* 包含下列其中一项或多项: • COMMIT • ROLLBACK • 所连接的数据库服务器未能准备的 SQL 语句 • 包含语法错误的语句
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	*szSqlStr* 是空指针。 自变量 *cbSqlStr* 小于 1，但不等于 SQL_NTS。
HY013 *****	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

注: 并非所有 DBMS 在进行准备时都报告上述所有诊断消息。因此，应用程序在调用 SQLExecute() 时还必须能够处理这些情况。

示例

有关以下示例中使用的 check_error、initialize 和 terminate 函数的列表，请参考第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』。

```
/************************************************************************
** file = prepare.c
**
** Example of preparing then repeatedly executing an SQL statement.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLTransact          SQLError
**        SQLPrepare           SQLSetParam
**        SQLExecute
************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

SQLPrepare

```
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  rc);

/*****************************************************************
** main
** - initialize
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt;
     SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = ?";
     SQLCHAR    deptname[15],
                location[14],
                division[11];

     SQLINTEGER rlength,
                plength;

        rc = SQLAllocStmt(hdbc, &hstmt);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        /* prepare statement for multiple use */
        rc = SQLPrepare(hstmt, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        /* bind division to parameter marker in sqlstmt */
        rc = SQLSetParam(hstmt, 1, SQL_CHAR, SQL_CHAR, 10, 10, division,
                    &plength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        /* bind deptname to first column in the result set */
        rc = SQLBindCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15,
                        &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);
        rc = SQLBindCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14,
                         &rlength);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);
```

```
        printf("\nEnter Division Name or 'q' to quit:\n");
        printf("(Eastern, Western, Midwest, Corporate)\n");
        gets(division);
        plength = SQL_NTS;

        while(division[0] != 'q')
        {
            rc = SQLExecute(hstmt);
            if (rc != SQL_SUCCESS )
                check_error (henv, hdbc, hstmt, rc);

            printf("Departments in %s Division:\n", division);
            printf("DEPTNAME       Location\n");
            printf("-------------- -------------\n");

            while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
            {
                printf("%-14.14s %-13.13s \n", deptname, location);
            }
            if (rc != SQL_NO_DATA_FOUND )
                check_error (henv, hdbc, hstmt, rc);
            SQLFreeStmt(hstmt, SQL_CLOSE);
            printf("\nEnter Division Name or 'q' to quit:\n");
            printf("(Eastern, Western, Midwest, Corporate)\n");
            gets(division);
        }
    }

    rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
}/* end main */
```

参考

- 第 58 页的『SQLColAttributes — 列属性』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 180 页的『SQLNumResultCols — 获取结果列数』

SQLPrimaryKeys — 获取表的主键列

用途

SQLPrimaryKeys() 返回组成表的主键的列的名称列表。将在 SQL 结果集中返回此信息，可以使用那些用来处理由查询生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN   SQLPrimaryKeys   (SQLHSTMT          StatementHandle,
                              SQLCHAR           *CatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *SchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *TableName,
                              SQLSMALLINT       NameLength3);
```

函数自变量

表 131. SQLPrimaryKeys 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLCHAR *	CatalogName	输入	由三部分组成的表名的目录限定符。 此项必须是空指针或长度为零的字符串。
SQLSMALLINT	NameLength1	输入	*CatalogName* 的长度
SQLCHAR *	SchemaName	输入	表名的模式限定符。
SQLSMALLINT	NameLength2	输入	*SchemaName* 的长度
SQLCHAR *	TableName	输入	表名。
SQLSMALLINT	NameLength3	输入	*TableName* 的长度

用法

SQLPrimaryKeys() 返回单个表中的主键列。不能使用搜索模式来指定模式限定符或表名。

结果集包含表 132 中列示的那些列，并且按 TABLE_CAT、TABLE_SCHEM、TABLE_NAME 和 ORDINAL_POSITION 排序。

由于在许多情况下调用 SQLPrimaryKeys() 都意味着要对系统目录进行复杂的查询（这样的成本很高），所以，您应该有节制地使用这些调用，并且应该保存结果而不是重复地进行调用。

虽然在将来的发行版中可能会添加新列，并可能会更改现有列的名称，但当前列的位置不会更改。

表 132. SQLPrimaryKeys 返回的列

列号 / 列名	数据类型	描述
1 TABLE_CAT	VARCHAR(128)	当前服务器。
2 TABLE_SCHEM	VARCHAR(128)	包含 TABLE_NAME 的模式的名称。
3 TABLE_NAME	VARCHAR(128) 非空	指定的表的名称。

表 132. SQLPrimaryKeys 返回的列 (续)

列号 / 列名	数据类型	描述
4 COLUMN_NAME	VARCHAR(128) 非空	主键列名。
5 ORDINAL_POSITION	SMALLINT 非空	主键中的列序号，从 1 开始。
6 PK_NAME	VARCHAR(128)	主键标识符。如果不适用于数据源，则为空。

注: DB2 UDB CLI 使用的列名遵循 X/Open CLI CAE 规范样式。列类型、内容和次序与 ODBC 中对 `SQLPrimaryKeys()` 结果集定义的那些列类型、内容和次序完全相同。

如果指定的表不包含主键，则返回空的结果集。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 133. SQLPrimaryKeys SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	已在语句句柄上打开了游标。
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY008	已将操作取消	
HY010	函数顺序错误	在执行数据（`SQLParamData()` 和 `SQLPutData()`）操作中调用了此函数。
HY014	没有更多的句柄	由于内部资源不足，所以 DB2 UDB CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HYC00	驱动程序不具有能力	DB2 UDB CLI 不支持将 *catalog* 用作表名的限定符。
HYT00	超时到期	

限制

无。

参考

- 第 105 页的『SQLForeignKeys — 获取外键列的列表』
- 第 231 页的『SQLStatistics — 获取基本表的索引和统计信息』

SQLProcedureColumns — 获取过程的输入 / 输出参数信息

用途

SQLProcedureColumns() 返回与某个过程相关联的输入和输出参数的列表。将在 SQL 结果集中返回此信息，可以使用那些用来处理由查询生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLProcedureColumns(SQLHSTMT          StatementHandle,
                              SQLCHAR           *CatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *SchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *ProcName,
                              SQLSMALLINT       NameLength3,
                              SQLCHAR           *ColumnName,
                              SQLSMALLINT       NameLength4);
```

函数自变量

表 134. SQLProcedureColumns 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLCHAR *	CatalogName	输入	由三部分组成的过程名的目录限定符。 这必须是空指针或长度为零的字符串。
SQLSMALLINT	NameLength1	输入	*CatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	SchemaName	输入	缓冲区，它可能包含*模式值*，以通过模式名限定结果集。 对于 DB2 UDB zOS 和 OS/390 版 V4.1，所有存储过程都在一个模式中；*SchemaName* 自变量的唯一可接受的值是空指针。对于 DB2 通用数据库™，*SchemaName* 可包含有效的模式值。
SQLSMALLINT	NameLength2	输入	*SchemaName* 的长度
SQLCHAR *	ProcName	输入	缓冲区，它可能包含*模式值*，以通过过程名限定结果集。
SQLSMALLINT	NameLength3	输入	*ProcName* 的长度
SQLCHAR *	ColumnName	输入	缓冲区，它可能包含*模式值*，以通过参数名限定结果集。此自变量将用来对已通过对 ProcName 或 SchemaName 指定非空值进行限制的结果集进行进一步的限定。
SQLSMALLINT	NameLength4	输入	*ColumnName* 的长度

用法

DB2 UDB CLI 将返回关于与存储过程相关联的输入、输出和输入输出参数的信息，但不能返回有关所返回的任何结果集的描述符信息的信息。

SQLProcedureColumns() 在一个结果集中返回信息，并且按 PROCEDURE_CAT、PROCEDURE_SCHEM、PROCEDURE_NAME 和 COLUMN_TYPE 排序。表 135 列示了结果集中的列。应用程序应该了解，在将来的发行版中可能会在最后一列之后定义新的列。

由于在许多情况下调用 SQLProcedureColumns() 都意味着要对系统目录进行复杂的查询（这样的成本很高），所以，您应该有节制地使用这些调用，并且应该保存结果而不是重复地进行调用。

表 135. SQLProcedureColumns 返回的列

列号 / 列名	数据类型	描述
1 PROCEDURE_CAT	VARCHAR(128)	当前服务器。
2 PROCEDURE_SCHEM	VARCHAR(128)	包含 PROCEDURE_NAME 的模式的名称。
3 PROCEDURE_NAME	VARCHAR(128)	过程的名称。
4 COLUMN_NAME	VARCHAR(128)	参数的名称。
5 COLUMN_TYPE	SMALLINT 非空	标识与此行相关联的类型信息。值可以是: • SQL_PARAM_TYPE_UNKNOWN — 参数类型未知。 **注**: 不返回这个值。 • SQL_PARAM_INPUT — 此参数是输入参数。 • SQL_PARAM_INPUT_OUTPUT — 此参数是输入 / 输出参数。 • SQL_PARAM_OUTPUT — 此参数是输出参数。 • SQL_RETURN_VALUE — 过程列是过程的返回值。 **注**: 不返回这个值。 • SQL_RESULT_COL — 此参数实际上是结果集中的某一列。 **注**: 不返回这个值。
6 DATA_TYPE	SMALLINT 非空	SQL 数据类型。
7 TYPE_NAME	VARCHAR(128) 非空	表示与 DATA_TYPE 相对应的数据类型的名称的字符串。
8 COLUMN_SIZE	INTEGER	如果 DATA_TYPE 列值指示字符或二进制字符串，则此列包含最大长度（字节）；如果是图形（DBCS）字符串，则这是参数的双字节字符数。 对于日期、时间和时间戳记数据类型，这是转换为字符后显示值所必需的总字节数。 对于数字数据类型，这是总位数或列中允许的总位数，这要视结果集中的 NUM_PREC_RADIX 列中的值而定。
9 BUFFER_LENGTH	INTEGER	如果在 SQLBindCol()、SQLGetData() 和 SQLBindParameter() 调用上指定了 SQL_C_DEFAULT，则此项是用于存储此参数中的数据的相关联 C 缓冲区的最大字节数。此长度不包括任何空终止符。对于精确数字数据类型，将小数和符号计入长度。
10 DECIMAL_DIGITS	SMALLINT	参数的标度。对于标度不适用的数据类型，将返回空。

SQLProcedureColumns

表 135. SQLProcedureColumns 返回的列 (续)

列号 / 列名	数据类型	描述
11 NUM_PREC_RADIX	SMALLINT	10、2 或空。如果 DATA_TYPE 是近似数字数据类型，则此列包含值 2，于是 COLUMN_SIZE 列包含参数中允许的位数。 如果 DATA_TYPE 是精确数字数据类型，则此列包含值 10，并且 COLUMN_SIZE 和 DECIMAL_DIGITS 列包含参数所允许的小数位数。 对于数字数据类型，DBMS 可以返回值为 10 或 2 的 NUM_PREC_RADIX。 对于基数不适用的数据类型，将返回空。
12 NULLABLE	VARCHAR(3)	如果参数不接受空值，则此列为 “NO”。 如果参数接受空值，则此列为 “YES”。
13 REMARKS	VARCHAR(254)	可能包含关于参数的描述性信息。
14 COLUMN_DEF	VARCHAR	列的缺省值。 如果指定了空作为缺省值，则此列将是空字，并且不用引号括起来。如果不进行截断就不能表示缺省值，则此列包含 TRUNCATED，并且不用单引号括起来。如果没有指定缺省值，则此列为空。 除非 COLUMN_DEF 包含值 TRUNCATED，否则可使用它的值来生成新的列定义。
15 SQL_DATA_TYPE	SMALLINT 非空	当 SQL 数据类型出现在描述符的 SQL_DESC_TYPE 字段中时它所具有的值。除用于日期时间数据类型之外，此列与 DATA_TYPE 列相同（DB2 UDB CLI 不支持时间间隔数据类型）。 对于日期时间数据类型，结果集中的 SQL_DATA_TYPE 字段将是 SQL_DATETIME，并且 SQL_DATETIME_SUB 字段将返回特定日期时间数据类型（SQL_CODE_DATE、SQL_CODE_TIME 或 SQL_CODE_TIMESTAMP）的子代码。
16 SQL_DATETIME_SUB	SMALLINT	日期时间数据类型的子类型代码。对于所有其它数据类型，此列包含空（包括 DB2 UDB CLI 不支持的时间间隔数据类型）。
17 CHAR_OCTET_LENGTH	INTEGER	字符数据类型列的最大长度（字节）。对于所有其它数据类型，此列返回空。
18 ORDINAL_POSITION	INTEGER 非空	包含此结果集中的 COLUMN_NAME 给出的参数的序数位置。这是在 CALL 语句上提供自变量时要采用的序数位置。最左边的自变量的序数位置为 1。

表 135. SQLProcedureColumns 返回的列 (续)

列号 / 列名	数据类型	描述
19 IS_NULLABLE	VARCHAR	• 如果列不能包含空，则为 “NO”。 • 如果列可以包含空，则为 “YES”。 • 如果未知可空性，则是长度为零的字符串。 遵循 ISO 规则来确定可空性。 与 ISO SQL 相符的 DBMS 不能返回空字符串。 对此列返回的值与对 NULLABLE 列返回的值不同。(参见 NULLABLE 列的描述。)

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 136. SQLProcedureColumns SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	已在语句句柄上打开了游标。
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
42601	PARMLIST 语法错误	存储过程目录表中的 PARMLIST 值包含语法错误。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY008	已将操作取消	
HY010	函数顺序错误	
HY014	没有更多的句柄	由于内部资源不足，所以 DB2 UDB CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HYC00	驱动程序不具有能力	DB2 UDB CLI 不支持将 *catalog* 用作过程名的限定符。 所连接的服务器不支持将 *schema* 用作过程名的限定符。
HYT00	超时到期	

限制

`SQLProcedureColumns()` 不返回关于可能从存储过程返回的结果集的属性的信息。

如果应用程序连接到未提供存储过程目录支持或未提供存储过程支持的 DB2 服务器，则 `SQLProcedureColumns()` 将返回空结果集。

示例

```
/* From CLI sample proccols.c */
/* ... */

    printf("Enter Procedure Schema Name Search Pattern:\n");
    gets((char *)proc_schem.s);

    printf("Enter Procedure Name Search Pattern:\n");
    gets((char *)proc_name.s);

    rc = SQLProcedureColumns(hstmt, NULL, 0, proc_schem.s, SQL_NTS,
                             proc_name.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) proc_schem.s, 129,
                    &proc_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) proc_name.s, 129,
                    &proc_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) column_name.s, 129,
                    &column_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 5, SQL_C_SHORT, (SQLPOINTER) &arg_type,
                    0, &arg_type_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) type_name.s, 129,
                    &type_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 8, SQL_C_LONG, (SQLPOINTER) & length,
                    0, &length_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 10, SQL_C_SHORT, (SQLPOINTER) &scale,
                    0, &scale_ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 13, SQL_C_CHAR, (SQLPOINTER) remarks.s, 255,
                    &remarks.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    /* Fetch each row, and display */
    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
        sprintf((char *)cur_name, "%s.%s", proc_schem.s, proc_name.s);
        if (strcmp((char *)cur_name, (char *)pre_name) != 0) {
            printf("\n%s\n", cur_name);
        }
        strcpy((char *)pre_name, (char *)cur_name);
        printf("   %s", column_name.s);
        switch (arg_type)
        { case SQL_PARAM_INPUT : printf(", Input"); break;
          case SQL_PARAM_OUTPUT : printf(", Output"); break;
          case SQL_PARAM_INPUT_OUTPUT : printf(", Input_Output"); break;
        }
        printf(", %s", type_name.s);
        printf(" (%ld", length);
        if (scale_ind != SQL_NULL_DATA) {
            printf(", %d)\n", scale);
        } else {
            printf(")\n");
        }
```

```
        if (remarks.ind > 0 ) {
            printf("(remarks), %s)\n", remarks.s);
        }
    }                           /* endwhile */
```

参考

- 第 198 页的『 SQLProcedures — 获取过程名列表 』

SQLProcedures — 获取过程名列表

用途

SQLProcedures() 返回已在服务器上注册并且与所指定的搜索模式相匹配的过程名的列表。

将在 SQL 结果集中返回此信息，可以使用那些用来处理由查询生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN   SQLProcedures    (SQLHSTMT          StatementHandle,
                              SQLCHAR           *CatalogName,
                              SQLSMALLINT       NameLength1,
                              SQLCHAR           *SchemaName,
                              SQLSMALLINT       NameLength2,
                              SQLCHAR           *ProcName,
                              SQLSMALLINT       NameLength3);
```

函数自变量

表 137. SQLTables 自变量

数据类型	自变量	用途	描述
SQLHSTMT	StatementHandle	输入	语句句柄。
SQLCHAR *	CatalogName	输入	由三部分组成的过程名的目录限定符。 这必须是空指针或长度为零的字符串。
SQLSMALLINT	NameLength1	输入	*CatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	SchemaName	输入	缓冲区，它可能包含*模式值*，以通过模式名限定结果集。 对于 DB2 UDB zOS 和 OS/390 版 V4.1，所有存储过程都在一个模式中；*SchemaName* 自变量的唯一可接受的值是空指针。对于 DB2 通用数据库，*SchemaName* 可包含有效的模式值。
SQLSMALLINT	NameLength2	输入	*SchemaName* 的长度。
SQLCHAR *	ProcName	输入	缓冲区，它可能包含*模式值*，以通过过程名限定结果集。
SQLSMALLINT	NameLength3	输入	*ProcName* 的长度。

用法

SQLProcedures() 返回的结果集包含表 138 中列示的列，并且具有所给出的次序。各行按 PROCEDURE_CAT、PROCEDURE_SCHEMA 和 PROCEDURE_NAME 排序。

由于在许多情况下调用 SQLProcedures() 都意味着要对系统目录进行复杂的查询（这样的成本很高），所以，请有节制地使用这些调用，并且保存结果而不是重复地进行调用。

虽然在将来的发行版中可能会添加新列，并可能会更改现有列的名称，但当前列的位置不会更改。

表 138. SQLProcedures 返回的列

1	PROCEDURE_CAT	VARCHAR(128)	当前服务器。

表 138. SQLProcedures 返回的列 (续)

2	PROCEDURE_SCHEM	VARCHAR(128)	包含 PROCEDURE_NAME 的模式的名称。
3	PROCEDURE_NAME	VARCHAR(128) 非空	过程的名称。
4	NUM_INPUT_PARAMS	INTEGER 非空	输入参数的数目。 不应该使用此列，它是为供 ODBC 将来使用而保留的。 在版本 5 之前的 DB2 UDB CLI 版本中使用了此列。为了向后兼容，可以将其与旧的 DB2CLI.PROCEDURES 伪目录表配合使用（通过设置“PATCH1 CLI/ODBC 配置”关键字）。
5	NUM_OUTPUT_PARAMS	INTEGER 非空	输出参数的数目。 不应该使用此列，它是为供 ODBC 将来使用而保留的。 在版本 5 之前的 DB2 UDB CLI 版本中使用了此列。为了向后兼容，可以将其与旧的 DB2CLI.PROCEDURES 伪目录表配合使用（通过设置“PATCH1 CLI/ODBC 配置”关键字）。
6	NUM_RESULT_SETS	INTEGER 非空	过程返回的结果集的数目。 不应该使用此列，它是为供 ODBC 将来使用而保留的。 在版本 5 之前的 DB2 UDB CLI 版本中使用了此列。为了向后兼容，可以将其与旧的 DB2CLI.PROCEDURES 伪目录表配合使用（通过设置“PATCH1 CLI/ODBC 配置”关键字）。
7	REMARKS	VARCHAR(254)	包含关于过程的描述性信息。

注: DB2 UDB CLI 使用的列名遵循 X/Open CLI CAE 规范样式。列类型、内容和次序与 ODBC 中对 `SQLProcedures()` 结果集定义的那些列类型、内容和次序完全相同。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

错误状态

表 139. SQLProcedures SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	已在语句句柄上打开了游标。
40003 **08**S01	通信链路故障	在函数完成之前，应用程序与数据源之间的通信链路发生故障。
HY001	内存分配失败	DB2 UDB CLI 无法分配支持此函数的执行或完成所必需的内存。
HY008	已将操作取消	
HY010	函数顺序错误	
HY014	没有更多的句柄	由于内部资源不足，所以 DB2 UDB CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。

SQLProcedures

表 139. SQLProcedures SQLSTATE (续)

SQLSTATE	描述	说明
HYC00	驱动程序不具有能力	DB2 UDB CLI 不支持将 *catalog* 用作过程名的限定符。 所连接的服务器不支持将 schema 用作过程名的限定符。
HYT00	超时到期	

限制

如果应用程序连接到未提供存储过程目录支持或未提供存储过程支持的 DB2 服务器，则 SQLProcedureColumns() 将返回空结果集。

示例

```
/* From CLI sample procs.c */
/* ... */

    printf("Enter Procedure Schema Name Search Pattern:\n");
    gets((char *)proc_schem.s);

    rc = SQLProcedures(hstmt, NULL, 0, proc_schem.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) proc_schem.s, 129,
                    &proc_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) proc_name.s, 129,
                    &proc_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) remarks.s, 255,
                    &remarks.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    printf("PROCEDURE SCHEMA          PROCEDURE NAME            \n");
    printf("------------------------ ------------------------ \n");
    /* Fetch each row, and display */
    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
        printf("%-25s %-25s\n", proc_schem.s, proc_name.s);
        if (remarks.ind != SQL_NULL_DATA) {
            printf("  (Remarks) %s\n", remarks.s);
        }
    }                           /* endwhile */
```

参考

- 第 192 页的『SQLProcedureColumns — 获取过程的输入／输出参数信息』

SQLPutData — 传送参数的数据值

用途

在返回 SQL_NEED_DATA 的 SQLParamData() 调用之后调用 SQLPutData() 来提供参数数据值。此函数可用来以分块方式发送大参数值。

语法

```
SQLRETURN SQLPutData  (SQLHSTMT     hstmt,
                       SQLPOINTER   rgbValue,
                       SQLINTEGER   cbValue);
```

函数自变量

表 140. SQLPutData 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLPOINTER	*rgbValue*	输入	指向参数的实际数据或部分数据的指针。此数据必须具有应用程序在指定参数时所使用的 SQLBindParam() 调用中指定的格式。
SQLINTEGER	*cbValue*	输入	*rgbValue* 的长度。指定在 SQLPutData() 调用中发送的数据量。 对于给定的参数，不同的调用可以使用不同的数据量。应用程序还可以对 *cbValue* 指定 SQL_NTS 或 SQL_NULL_DATA。 对于所有日期、时间、时间戳记数据类型以及所有除 SQL_NUMERIC 和 SQL_DECIMAL 之外的数字数据类型，忽略 *cbValue*。 对于 C 缓冲区类型是 SQL_CHAR 或 SQL_BINARY 的情况，或者，如果指定了 SQL_DEFAULT 来作为 C 缓冲区类型，并且 C 缓冲区类型缺省值是 SQL_CHAR 或 SQL_BINARY，则此自变量是 *rgbValue* 缓冲区中的数据的字节数。

用法

应用程序在对处于 SQL_NEED_DATA 状态的语句调用 SQLParamData() 之后调用 SQLPutData() 来为 SQL_DATA_AT_EXEC 参数提供数据值。可通过重复地调用 SQLPutData() 来以分块方式发送长数据。在发送参数的所有数据块之后，应用程序再次调用 SQLParamData()。SQLParamData() 将继续处理下一个 SQL_DATA_AT_EXEC 参数，或者，如果所有参数都已具有数据值，则执行语句。

不能对定长参数多次调用 SQLPutData()。

如果输入数据是字符或二进制数据，则在调用 SQLPutData() 之后，合法的函数调用只有 SQLParamData()、SQLCancel() 或另一个 SQLPutData()。对于 SQLParamData()，所有其它使用此语句句柄的

函数调用都将失败。另外，对于引用 *hstmt* 的父 *hdbc* 的函数调用，如果它们涉及更改该连接的任何属性或状态，则它们全都会失败。要获取这些函数的列表，请查看第 182 页的『SQLParamData — 获取下一个需要其数据值的参数』的用法部分。

如果对单个参数执行的一个或多个 SQLPutData() 调用导致 SQL_SUCCESS，则在 *cbValue* 设置为 SQL_NULL_DATA 的情况下尝试对同一参数调用 SQLPutData() 将导致错误，并且 SQLSTATE 为 **HY**011。此错误不会导致状态更改；语句句柄仍处于*需要数据*状态，并且应用程序可以继续发送参数数据。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

下面的一些诊断情况可能会在最后进行 SQLParamData() 调用时报告，而不是在调用 SQLPutData() 时报告。

表 141. SQLPutData SQLSTATE

SQLSTATE	描述	说明
22001	数据太多	SQLPutData() 提供给当前参数的数据的大小超出参数的大小。将忽略上次调用 SQLPutData() 时提供的数据。
01004	已截断数据	已截断为数字参数发送的数据，没有丢失有效位。 已截断为日期或时间列发送的时间戳记数据。 函数将返回 SQL_SUCCESS_WITH_INFO。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	自变量 *rgbValue* 是空指针。 自变量 *rgbValue* 不是空指针，而自变量 *cbValue* 小于 0，但不等于 SQL_NTS 或 SQL_NULL_DATA。
HY010	函数顺序错误	语句句柄 *hstmt* 必须处于**需要数据**状态，并且必须已通过前一次 SQLParamData() 调用定位在 SQL_DATA_AT_EXEC 参数上。

SQLReleaseEnv — 释放所有环境资源

用途

SQLReleaseEnv() 使环境句柄无效并将其释放。将释放与该环境句柄相关联的所有 DB2 UDB CLI 资源。

在调用此函数之前，必须调用 SQLFreeConnect()。

此函数是应用程序在终止之前所需执行的最后一个 DB2 UDB CLI 步骤。

语法

```
SQLRETURN SQLReleaseEnv (SQLHENV    henv);
```

函数自变量

表 142. SQLReleaseEnv 自变量

数据类型	自变量	用途	描述
SQLHENV	*henv*	输入	环境句柄

用法

如果在仍然存在有效连接句柄的情况下调用此函数，则将返回 SQL_ERROR，并且环境句柄将保持有效。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 143. SQLReleaseEnv SQLSTATE

SQLSTATE	描述	说明
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY010	函数顺序错误	存在处于已分配或已连接状态的 *hdbc*。在调用 SQLReleaseEnv 之前，请对该 *hdbc* 调用 SQLDisconnect 和 SQLFreeConnect。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

请参考 SQLAllocEnv()（第 28 页的『示例』）。

参考

- 第 110 页的『SQLFreeConnect — 释放连接句柄』

SQLRowCount — 获取行计数

用途

SQLRowCount() 返回对一个表或基于该表的视图执行的 UPDATE、INSERT 或 DELETE 语句所影响的该表中的行数。

在调用此函数之前，必须调用 SQLExecute() 或 SQLExecDirect()。

语法

```
SQLRETURN SQLRowCount (SQLHSTMT      hstmt,
                       SQLINTEGER    *pcrow);
```

函数自变量

表 144. SQLRowCount 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLINTEGER *	*pcrow*	输出	指向一个位置的指针，受影响的行数存储在该位置中。

用法

如果输入语句句柄所引用的上次执行的语句不是 UPDATE、INSERT 或 DELETE 语句，或者它没有执行成功，则此函数将把 *pcrow* 的内容设置为 0。

其它表中任何可能受该语句影响（例如级联删除）的行不包括在此计数中。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 145. SQLRowCount SQLSTATE

SQLSTATE	描述	说明
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	*pcrow* 是空指针
HY010	函数顺序错误	在对 *hstmt* 调用 SQLExecute 或 SQLExecDirect 之前调用了此函数。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 180 页的『SQLNumResultCols — 获取结果列数』

SQLSetConnectAttr — 设置连接属性

用途

SQLSetConnectAttr() 设置特定连接的连接属性。

语法

```
SQLRETURN SQLSetConnectAttr  (SQLHDBC     hdbc,
                              SQLINTEGER  fAttr,
                              SQLPOINTER  vParam,
                              SQLINTEGER  sLen);
```

函数自变量

表 146. SQLSetConnectAttr 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄
SQLINTEGER	*fAttr*	输入	要设置的连接属性，有关更多信息，请参考表 147。
SQLPOINTER	*vParam*	输入	与 *fAttr* 相关联的值。根据选项的不同，这可以是指向 32 位整数值的指针，也可以是字符串。
SQLINTEGER	*sLen*	输入	如果输入值是字符串的话，此自变量就是它的长度；否则不使用此自变量。

用法

通过 SQLSetConnectAttr() 设置的所有连接和语句选项都将持续到调用 SQLFreeConnect() 或下次调用 SQLSetConnectAttr() 时为止。

通过 *vParam* 设置的信息的格式取决于所指定的 *fAttr*。选项信息可以是 32 位整数，也可以是指向以空终止的字符串的指针。

表 147. 连接选项

fAttr	内容
SQL_ATTR_AUTOCOMMIT	32 位的值，它设置连接的落实行为。可能的值列示如下: • SQL_TRUE — 在执行 SQL 语句时自动落实每个 SQL 语句。 • SQL_FALSE — 不自动落实 SQL 语句。如果在具有落实控制的环境下运行，则必须使用 SQLEndTran() 或 SQLTransact() 来显式地落实或回滚更改。

表 147. 连接选项 (续)

fAttr	内容
SQL_ATTR_COMMIT 或 SQL_TXN_ISOLATION	32 位的值，它设置 *hdbc* 所引用的当前连接的事务隔离级别。DB2 UDB CLI 接受下列值，但每个服务器都可能只支持这些隔离级别的其中某一些: • SQL_TXN_NO_COMMIT — 未使用落实控制。 • SQL_TXN_READ_UNCOMMITTED — 脏读取、不可重复读取和幻象读取是有可能的。 • SQL_TXN_READ_COMMITTED — 脏读取是不可能的。不可重复读取和幻象读取是有可能的。 • SQL_TXN_REPEATABLE_READ — 脏读取和不可重复读取是不可能的。幻象读取是有可能的。 • SQL_TXN_SERIALIZABLE — 事务是可序列化的。脏读取、不可重复读取和幻象都是不可能的。 在 IBM 术语中: • SQL_TXN_READ_UNCOMMITTED 是指“未落实的读取”； • SQL_TXN_READ_COMMITTED 是指“游标稳定性”； • SQL_TXN_REPEATABLE_READ 是指“读取稳定性”； • SQL_TXN_SERIALIZABLE 是指“可重复读取”。 有关“隔离级别”的详细说明，请参考 IBM SQL Reference。 应该在调用 `SQLConnect()` 之前设置 SQL_ATTR_COMMIT 属性。如果在建立连接之后更改了值，并且该连接是与远程数据源的连接，则在下一次对连接句柄成功调用 `SQLConnect()` 之前，此更改不会生效。
SQL_ATTR_DATE_FMT	32 位的整数值，这个值可以是: • SQL_FMT_ISO — 使用“国际标准化组织”（ISO）日期格式 yyyy-mm-dd。这是缺省值。 • SQL_FMT_USA — 使用美国日期格式 mm/dd/yyyy。 • SQL_FMT_EUR — 使用欧洲日期格式 dd.mm.yyyy。 • SQL_FMT_JIS — 使用“日本工业标准”日期格式 yyyy-mm-dd。 • SQL_FMT_MDY — 使用日期格式 mm/dd/yyyy。 • SQL_FMT_DMY — 使用日期格式 dd/mm/yyyy。 • SQL_FMT_YMD — 使用日期格式 yy/mm/dd。 • SQL_FMT_JUL — 使用儒略日期格式 yy/ddd。 • SQL_FMT_JOB — 使用作业缺省值。

表 147. 连接选项 (续)

fAttr	内容
SQL_ATTR_DATE_SEP	32 位的整数值，这个值可以是: • SQL_SEP_SLASH — 使用斜杠（/）作为日期分隔符。这是缺省值。 • SQL_SEP_DASH — 使用破折号（-）作为日期分隔符。 • SQL_SEP_PERIOD — 使用句点（.）作为日期分隔符。 • SQL_SEP_COMMA — 使用逗号（,）作为日期分隔符。 • SQL_SEP_BLANK — 使用空格作为日期分隔符。 • SQL_SEP_JOB — 使用作业缺省值。
SQL_ATTR_DBC_DEFAULT_LIB	一个字符值，这个值指示将用于解析未限定文件引用的缺省库。如果连接正在使用系统命名方式，则此项无效。
SQL_ATTR_DBC_SYS_NAMING	32 位的整数值，这个值可以是: • SQL_TRUE — DB2 UDB CLI 使用 iSeries 系统命名方式。使用斜杠（/）定界符来限定文件。使用作业的库列表来解析未限定的文件。 • SQL_FALSE — DB2 UDB CLI 使用缺省命名方式，即 SQL 命名。使用句点（.）定界符来限定文件。使用缺省库或当前用户标识来解析未限定的文件。
SQL_ATTR_DECIMAL_SEP	32 位的整数值，这个值可以是: • SQL_SEP_PERIOD — 使用句点（.）作为小数分隔符。这是缺省值。 • SQL_SEP_COMMA — 使用逗号（,）作为日期分隔符。 • SQL_SEP_JOB — 使用作业缺省值。
SQL_ATTR_EXTENDED_COL_INFO	32 位的整数值，这个值可以是: • SQL_TRUE — 可以在 SQLColAttributes() 上使用对此连接句柄分配的语句句柄来检索扩展的列信息，如“基本表”、“基本模式”、“基本列”和“标号”。 • SQL_FALSE — 不能在 SQLColAttributes() 函数上使用对此连接句柄分配的语句句柄来检索扩展的列信息。这是缺省值。
SQL_ATTR_TIME_FMT	32 位的整数值，这个值可以是: • SQL_FMT_ISO — 使用“国际标准化组织”（ISO）时间格式 hh.mm.ss。这是缺省值。 • SQL_FMT_USA — 使用美国时间格式 hh:mmxx，其中 xx 是 AM 或 PM。 • SQL_FMT_EUR — 使用欧洲时间格式 hh.mm.ss。 • SQL_FMT_JIS — 使用“日本工业标准”时间格式 hh:mm:ss。 • SQL_FMT_HMS — 使用 hh:mm:ss 格式。

表 147. 连接选项 (续)

fAttr	内容
SQL_ATTR_TIME_SEP	32 位的整数值，这个值可以是: • SQL_SEP_COLON — 使用冒号（:）作为时间分隔符。这是缺省值。 • SQL_SEP_PERIOD — 使用句点（.）作为时间分隔符。 • SQL_SEP_COMMA — 使用逗号（,）作为时间分隔符。 • SQL_SEP_BLANK — 使用空格作为时间分隔符。 • SQL_SEP_JOB — 使用作业缺省值。
SQL_SAVEPOINT_NAME	一个字符值，它指示 SQLEndTran() 要对函数 SQL_SAVEPOINT_NAME_ROLLBACK 或 SQL_SAVEPOINT_NAME_RELEASE 使用的保存点名称。
SQL_2ND_LEVEL_TEXT	32 位的整数值，这个值可以是: • SQL_TRUE — 通过调用 SQLError() 获取的错误文本将包含关于错误的完整文本描述。 • SQL_FALSE — 通过调用 SQLError() 获取的错误文本将只包含关于错误的第一个级别的描述。这是缺省值。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 148. SQLSetConnectAttr SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	给定 *fAttr* 值，对自变量 *vParam* 指定了无效的值。 指定了无效的 *fAttr* 值。

SQLSetConnectOption — 设置连接选项

用途

SQLSetConnectOption() 设置特定连接的连接属性。

语法

```
SQLRETURN SQLSetConnectOption  (SQLHDBC hdbc,
                                SQLSMALLINT fOption,
                                SQLPOINTER  vParam);
```

函数自变量

表 149. SQLSetConnectOption 自变量

数据类型	自变量	用途	描述
SQLHDBC	*hdbc*	输入	连接句柄
SQLSMALLINT	*fOption*	输入	要设置的连接选项，有关更多信息，请参考第 206 页的表 147。
SQLPOINTER	*vParam*	输入	与 *fOption* 相关联的值。根据选项的不同，这可以是指向 32 位整数值的指针，也可以是字符串。

用法

SQLSetConnectOption() 与 SQLSetConnectAttr() 提供相同的功能，支持这两个函数的目的都是为了提高兼容性。

通过 SQLSetConnectOption() 设置的所有连接和语句选项都将持续到调用 SQLFreeConnect() 或下次调用 SQLSetConnectOption() 时为止。

通过 *vParam* 设置的信息的格式取决于所指定的 *fOption*。选项信息可以是 32 位整数，也可以是指向以空终止的字符串的指针。

请参考第 206 页的表 147 以了解适当的连接选项。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 150. SQLSetConnectOption SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。

表 150. SQLSetConnectOption SQLSTATE (续)

SQLSTATE	描述	说明
HY009	自变量值无效	给定 *fOption* 值，对自变量 *vParam* 指定了无效的值。 指定了无效的 *fOption* 值。
HYC00	驱动程序不具有能力	DB2 UDB CLI 或服务器不支持指定的 *fOption*。 给定所指定的 *fOption* 值，不支持对自变量 *vParam* 指定的值。

SQLSetCursorName — 设置游标名

用途

SQLSetCursorName() 使游标名与语句句柄相关联。由于 DB2 UDB CLI 会在需要时隐式地生成游标名，所以此函数是可选的。

语法

```
SQLRETURN SQLSetCursorName (SQLHSTMT       hstmt,
                            SQLCHAR        *szCursor,
                            SQLSMALLINT    cbCursor);
```

函数自变量

表 151. SQLSetCursorName 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLCHAR *	*szCursor*	输入	游标名
SQLSMALLINT	*cbCursor*	输入	*szCursor* 自变量的内容的长度

用法

当准备或直接执行 SELECT 语句时，DB2 UDB CLI 总是生成并使用内部生成的游标名。SQLSetCursorName() 允许在 SQL 语句（定位型 UPDATE 或 DELETE）中使用应用程序定义的游标名。DB2 UDB CLI 将把此名称映射到内部名称。在生成内部名称之前，必须调用 SQLSetCursorName()。在删除语句句柄之前，该名称将保持与该句柄相关联。该名称在事务结束之后也会一直保持存在，但这时候可调用 SQLSetCursorName() 来对此语句句柄设置另一个名称。

游标名必须遵循下列规则:

- 连接内的所有游标名都必须是唯一的。
- 每个游标名的长度都必须小于或等于 18 个字节。任何将游标名设置为长于 18 个字节的尝试都会导致将该游标名截断为 18 个字节。（不生成警告。）
- 由于在 SQL 中将游标名视为标识符，所以它必须以英语字母（a-z 和 A-Z）开头，后跟数字（0-9）、英语字母或下划线字符（_）的任意组合。
- 除非将输入游标名括在双引号中，否则将从输入游标名中除去所有前导和尾部空格。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 152. SQLSetCursorName SQLSTATE

SQLSTATE	描述	说明
34000	游标名无效	自变量 *szCursor* 指定的游标名无效。游标名应该以“SQLCUR”或“SQL_CUR”开头，否则将违反驱动程序或数据源游标命名规则（必须以 a-z 或 A-Z 开头，后跟英语字母、数字或“_”字符的任意组合）。 自变量 *szCursor* 指定的游标名已存在。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	*szCursor* 是空指针。 自变量 *cbCursor* 小于 1，但不等于 SQL_NTS。
HY010	函数顺序错误	语句句柄未处于已分配状态。 在调用 `SQLSetCursorName()` 之前调用了 `SQLPrepare()` 或 `SQLExecDirect()`。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 123 页的『SQLGetCursorName — 获取游标名』

SQLSetDescField — 设置描述符字段

用途

SQLSetDescField() 设置描述符中的字段。SQLSetDescField() 是 SQLSetDescRec() 函数的更具扩展性的备用函数。

语法

```
SQLRETURN SQLSetDescField  (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLSMALLINT    fDescType,
                            SQLPOINTER     rgbDesc,
                            SQLINTEGER     bLen);
```

函数自变量

表 153. SQLSetDescField 自变量

数据类型	自变量	用途	描述
SQLHDESC	*hdesc*	输入	描述符句柄
SQLSMALLINT	*irec*	输入	要从中检索所指定的字段的记录的编号
SQLSMALLINT	*fDescType*	输入	参见表 154。
SQLPOINTER	*rgbDesc*	输入	指向缓冲区的指针
SQLINTEGER	*bLen*	输入	描述符缓冲区（*rgbDesc*）的长度

表 154. fDescType 描述符类型

描述符	类型	描述
SQL_DESC_COUNT	SMALLINT	设置描述符中的记录数。将忽略 *irec*。
SQL_DESC_TYPE	SMALLINT	设置 *irec* 的类型字段。
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	设置类型为 SQL_DATETIME 的记录的时间间隔代码
SQL_DESC_LENGTH	INTEGER	设置 *irec* 的长度字段。
SQL_DESC_PRECISION	SMALLINT	设置 *irec* 的精度字段。
SQL_DESC_SCALE	SMALLINT	设置 *irec* 的标度字段。
SQL_DESC_DATA_PTR	SQLPOINTER	设置 *irec* 的数据指针字段。
SQL_DESC_LENGTH_PTR	SQLPOINTER	设置 *irec* 的长度指针字段。
SQL_DESC_INDICATOR_PTR	SQLPOINTER	设置 *irec* 的指示符指针字段。

用法

SQLSetDescField() 不象 SQLSetDescRec() 那样需要整个自变量集合，相反，它指定您想要对特定描述符记录设置的属性。

虽然 SQLSetDescField() 允许将来进行扩展，但与 SQLSetDescRec() 相比，它要求对每个描述符记录进行更多的调用才能设置相同的信息。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 155. SQLGetDescField SQLSTATE

SQLSTATE	描述	说明
HY009	自变量值无效	对自变量 *fDescType* 或 *irec* 指定的值无效。 自变量 *rgbValue* 是空指针。
HY013 *****	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 186 页的『SQLPrepare — 准备语句』

SQLSetDescRec — 设置描述符记录

用途

SQLSetDescRec() 设置描述符记录的所有属性。SQLSetDescRec() 是 SQLDescField() 函数的更简洁的备用函数。

语法

```
SQLRETURN SQLSetDescRec (SQLHDESC       hdesc,
                         SQLSMALLINT    irec,
                         SQLSMALLINT    type,
                         SQLSMALLINT    subtype,
                         SQLINTEGER     length,
                         SQLSMALLINT    prec,
                         SQLSMALLINT    scale,
                         SQLPOINTER     data,
                         SQLINTEGER     *sLen,
                         SQLINTEGER    *indic);
```

函数自变量

表 156. SQLSetDescRec 自变量

数据类型	自变量	用途	描述
SQLDESC	*hdesc*	输入	描述符句柄
SQLSMALLINT	*irec*	输入	描述符内的记录号。
SQLSMALLINT	*type*	输入	记录的 TYPE 字段。
SQLSMALLINT	*subtype*	输入	TYPE 为 SQL_DATETIME 的记录的 DATETIME_INTERVAL_CODE 字段。
SQLINTEGER	*length*	输入	记录的 LENGTH 字段。
SQLSMALLINT	*prec*	输入	记录的 PRECISION 字段。
SQLSMALLINT	*scale*	输入	记录的 SCALE 字段。
SQLPOINTER	*data*	输入（延迟）	记录的 DATA_PTR 字段。
SQLINTEGER *	*sLen*	输入（延迟）	记录的 LENGTH_PTR 字段。
SQLINTEGER *	*indic*	输入（延迟）	记录的 INDICATOR_PTR 字段。

用法

通过调用 SQLSetDescRec()，将通过一次调用来设置描述符记录的所有字段。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 157. SQLSetDescRec SQLSTATE

SQLSTATE	描述	说明
HY009	自变量值无效	对自变量 *irec* 指定的值小于 1。 对另一个自变量指定了无效的值。
HY016	描述符无效	用于引用实现行描述符的描述符句柄。

参考

- 第 34 页的『SQLBindCol — 将列绑定到应用程序变量』
- 第 74 页的『SQLDescribeCol — 描述列属性』
- 第 91 页的『SQLExecDirect — 直接执行语句』
- 第 93 页的『SQLExecute — 执行语句』
- 第 186 页的『SQLPrepare — 准备语句』

SQLSetEnvAttr — 设置环境属性

用途

SQLSetEnvAttr() 设置当前环境的环境属性。

语法

```
SQLRETURN SQLSetEnvAttr  (SQLHENV      henv,
                          SQLINTEGER   Attribute,
                          SQLPOINTER   Value,
                          SQLINTEGER   StringLength);
```

函数自变量

表 158. SQLSetEnvAttr 自变量

数据类型	自变量	用途	描述
SQLHENV	*henv*	输入	环境句柄
SQLINTEGER	*Attribute*	输入	要设置的环境属性，有关更多信息，请参考表 159。
SQLPOINTER	*pValue*	输入	*Attribute* 的期望值。
SQLINTEGER	*StringLength*	输入	如果属性值是字符串，则此自变量是 *Value* 的长度（字节）；如果 *Attribute* 指示的不是字符串，则 DB2 UDB CLI 忽略 *StringLength*。一定不能是任何已处理的连接句柄，否则将发生 HY010 错误。

用法

表 159. 环境属性

属性	内容
SQL_ATTR_DATE_FMT	32 位的整数值，这个值可以是: • SQL_FMT_ISO — 使用“国际标准化组织”（ISO）日期格式 yyyy-mm-dd。这是缺省值。 • SQL_FMT_USA — 使用美国日期格式 mm/dd/yyyy。 • SQL_FMT_EUR — 使用欧洲日期格式 dd.mm.yyyy。 • SQL_FMT_JIS — 使用“日本工业标准”日期格式 yyyy-mm-dd。 • SQL_FMT_MDY — 使用日期格式 mm/dd/yyyy。 • SQL_FMT_DMY — 使用日期格式 dd/mm/yyyy。 • SQL_FMT_YMD — 使用日期格式 yy/mm/dd。 • SQL_FMT_JUL — 使用儒略日期格式 yy/ddd。 • SQL_FMT_JOB — 使用作业缺省值。

表 159. 环境属性 (续)

属性	内容
SQL_ATTR_DATE_SEP	32 位的整数值，这个值可以是: • SQL_SEP_SLASH — 使用斜杠（/）作为日期分隔符。这是缺省值。 • SQL_SEP_DASH — 使用破折号（-）作为日期分隔符。 • SQL_SEP_PERIOD — 使用句点（.）作为日期分隔符。 • SQL_SEP_COMMA — 使用逗号（,）作为日期分隔符。 • SQL_SEP_BLANK — 使用空格作为日期分隔符。 • SQL_SEP_JOB — 使用作业缺省值。
SQL_ATTR_DECIMAL_SEP	32 位的整数值，这个值可以是: • SQL_SEP_PERIOD — 使用句点（.）作为小数分隔符。这是缺省值。 • SQL_SEP_COMMA — 使用逗号（,）作为日期分隔符。 • SQL_SEP_JOB — 使用作业缺省值。
SQL_ATTR_DEFAULT_LIB	一个字符值，这个值指示将用于解析未限定文件引用的缺省库。如果环境正在使用系统命名方式，则此项无效。
SQL_ATTR_ENVHNDL_COUNTER	32 位的整数值，这个值可以是: • SQL_FALSE — DB2 CLI 不对分配环境句柄的次数进行计数。因此，第一个用于释放环境的调用将释放句柄以及所有相关联的资源。 • SQL_TRUE — DB2 CLI 维护一个计数器来记录分配环境句柄的次数。每次释放环境句柄时，此计数器都将递减。仅当计数器达到零时，DB2 CLI 才实际释放句柄以及所有相关联的资源。这允许使用 CLI 来对分配和释放 CLI 环境句柄的程序进行嵌套调用。
SQL_ATTR_ESCAPE_CHAR	字符值，它指示在 SQLColumns() 或 SQLTables() 中指定搜索模式时要使用的转义字符。
SQL_ATTR_FOR_FETCH_ONLY	32 位的整数值，这个值可以是: • SQL_TRUE — 游标是只读的，并且不能用于定位型更新或删除。这是缺省值。 • SQL_FALSE — 游标可用于定位型更新和删除。 还可以使用 SQLSetStmtAttr() 来对个别语句设置属性 SQL_ATTR_FOR_FETCH_ONLY。
SQL_ATTR_JOB_SORT_SEQUENCE	32 位的整数值，这个值可以是: • SQL_TRUE — DB2 UDB CLI 使用已经为作业设置的排序顺序。 • SQL_FALSE — DB2 UDB CLI 使用缺省排序顺序 *HEX。

表 159. 环境属性 (续)

属性	内容
SQL_ATTR_OUTPUT_NTS	32 位的整数值，这个值可以是: • SQL_TRUE — DB2 UDB CLI 使用空终止来指示输出字符串的长度。 • SQL_FALSE — DB2 UDB CLI 不使用空终止 受此属性影响的 CLI 函数是为该环境（以及为该环境下分配的任何连接）调用的具有字符串参数的所有函数。
SQL_ATTR_SERVER_MODE	32 位的整数值，这个值可以是: • SQL_FALSE — DB2 CLI 在同一个作业中处理所有连接的 SQL 语句。所有更改组成单个事务。这是缺省处理方式。 • SQL_TRUE — DB2 CLI 在单独的作业中处理每个连接的 SQL 语句。这允许与同一个数据源建立多个连接（可能对每个连接使用不同的用户标识）。它还将在每个连接句柄下所作的更改分隔到它自己的事务中。这允许落实或回滚每个连接句柄，而不会影响在其它连接句柄下所作的暂挂更改。有关更多信息，参见第 271 页的附录 D, 『以服务器方式运行 DB2 UDB CLI』。
SQL_ATTR_SYS_NAMING	32 位的整数值，这个值可以是: • SQL_TRUE — DB2 UDB CLI 使用 iSeries 系统命名方式。使用斜杠（/）定界符来限定文件。使用作业的库列表来解析未限定的文件。 • SQL_FALSE — DB2 UDB CLI 使用缺省命名方式，即 SQL 命名。使用句点（.）定界符来限定文件。使用缺省库或当前用户标识来解析未限定的文件。
SQL_ATTR_TIME_FMT	32 位的整数值，这个值可以是: • SQL_FMT_ISO — 使用“国际标准化组织”（ISO）时间格式 hh.mm.ss。这是缺省值。 • SQL_FMT_USA — 使用美国时间格式 hh:mmxx，其中 xx 是 AM 或 PM。 • SQL_FMT_EUR — 使用欧洲时间格式 hh.mm.ss。 • SQL_FMT_JIS — 使用“日本工业标准”时间格式 hh:mm:ss。 • SQL_FMT_HMS — 使用 hh:mm:ss 格式。
SQL_ATTR_TIME_SEP	32 位的整数值，这个值可以是: • SQL_SEP_COLON — 使用冒号（:）作为时间分隔符。这是缺省值。 • SQL_SEP_PERIOD — 使用句点（.）作为时间分隔符。 • SQL_SEP_COMMA — 使用逗号（,）作为时间分隔符。 • SQL_SEP_BLANK — 使用空格作为时间分隔符。 • SQL_SEP_JOB — 使用作业缺省值。

表 159. 环境属性 (续)

属性	内容
SQL_ATTR_UTF8	32 位的整数值，这个值可以是: • SQL_FALSE — 将字符数据视为具有作业缺省 CCSID。这是缺省值。 • SQL_TRUE — 将字符数据视为具有 UTF-8 CCSID（1208）。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 160. SQLSetEnvAttr SQLSTATE

SQLSTATE	描述	说明
HY009	参数值无效	DB2 UDB CLI 不支持指定的 *Attribute*。 给定所指定的 *Attribute* 值，不支持对自变量 *Value* 指定的值。 自变量 *pValue* 是空指针。
HY010	函数顺序错误	已分配连接句柄。

SQLSetParam — 设置参数

用途

SQLSetParam() 使应用程序变量与 SQL 语句中的参数标记相关联（将应用程序变量绑定到参数标记）。执行语句时，将把绑定的变量的内容发送至数据库服务器。此函数还用来指定任何必需的数据转换。

语法

```
SQLRETURN SQLSetParam (SQLHSTMT       hstmt,
                       SQLSMALLINT    ipar,
                       SQLSMALLINT    fCType,
                       SQLSMALLINT    fSqlType,
                       SQLINTEGER     cbParamDef,
                       SQLSMALLINT    ibScale,
                       SQLPOINTER     rgbValue,
                       SQLINTEGER     *pcbValue);
```

注: 有关此函数的描述，请参考第 44 页的『SQLBindParam — 将缓冲区绑定到参数标记』。那些函数是完全相同的，支持它们是为了提高兼容性。

SQLSetStmtAttr — 设置语句属性

用途

SQLSetStmtAttr() 设置特定语句句柄的属性。要为与某个连接句柄相关联的所有语句句柄设置选项，应用程序可以调用 SQLSetConnectOption()（有关其它详细信息，另请参考第 210 页的『SQLSetConnectOption — 设置连接选项』）。

语法

```
SQLRETURN SQLSetStmtAttr  (SQLHSTMT       hstmt,
                           SQLINTEGER     fAttr,
                           SQLPOINTER     vParam,
                           SQLINTEGER     sLen);
```

函数自变量

表 161. SQLSetStmtAttr 自变量

数据类型	自变量	用途	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLINTEGER	*fAttr*	输入	要设置的属性。有关可设置的语句属性的列表，请参考表 162。
SQLPOINTER	*vParam*	输入	与 *fAttr* 相关联的值。*vParam* 可以是 32 位整数值或字符串。
SQLINTEGER	*sLen*	输入	如果数据是字符串，则此自变量是数据的长度；否则不使用此自变量。

用法

在另一个 SQLSetStmtAttr() 调用更改 *hstmt* 的语句选项或通过使用 SQL_DROP 选项调用 SQLFreeStmt() 来删除该 *hstmt* 之前，那些语句选项将保持有效。使用 SQL_CLOSE、SQL_UNBIND 或 SQL_RESET_PARAMS 选项调用 SQLFreeStmt() 并不会重设语句选项。

通过 *vParam* 设置的信息的格式取决于所指定的 *fOption*。表 162 说明了每种信息的格式。

表 162. 语句属性

fAttr	内容
SQL_ATTR_APP_PARAM_DESC	*VParam* 必须是描述符句柄。在以后对语句句柄调用 SQLExecute() 和 SQLExecDirect() 时，指定的描述符将用作应用程序参数描述符。
SQL_ATTR_APP_ROW_DESC	*VParam* 必须是描述符句柄。在以后对语句句柄调用 SQLFetch() 时，指定的描述符将用作应用程序行描述符。
SQL_ATTR_CURSOR_HOLD	32 位的整数值，这个值指定为此语句句柄打开的游标是否应该挂起。 • SQL_FALSE — 在执行落实或回滚操作时，将把此语句句柄的打开的游标关闭。这是缺省值。 • SQL_TRUE — 在执行落实或回滚操作时，不会将此语句句柄的打开的游标关闭。

表 162. 语句属性 (续)

fAttr	内容
SQL_ATTR_CURSOR_SCROLLABLE	32 位的整数值，这个值指定为此语句句柄打开的游标是否应该可滚动。 • SQL_FALSE — 游标不可滚动，不能再次对它们使用 SQLFetchScroll()。这是缺省值。 • SQL_TRUE — 游标可滚动。可使用 SQLFetchScroll() 来从这些游标检索数据。
SQL_ATTR_CURSOR_TYPE	32 位的整数值，这个值指定为此语句句柄打开的游标的行为。 • SQL_CURSOR_FORWARD_ONLY — 游标不可滚动，不能再次对它们使用 SQLFetchScroll()。这是缺省值。 • SQL_DYNAMIC — 游标可滚动。可使用 SQLFetchScroll() 来从这些游标检索数据。
SQL_ATTR_EXTENDED_COL_INFO	32 位的整数值，这个值指定为此语句句柄打开的游标是否应该提供扩展的列信息。 • SQL_FALSE — 不能在 SQLColAttributes() 函数上使用此语句句柄来检索扩展的列信息。这是缺省值。在语句级别上设置此属性将覆盖此属性的连接级别设置。 • SQL_TRUE — 可以在 SQLColAttributes() 上使用此语句句柄来检索扩展的列信息，如“基本表”、“基本模式”、“基本列”和“标号”。
SQL_ATTR_FOR_FETCH_ONLY	32 位的整数值，这个值指定为此语句句柄打开的游标是否应该是只读的。 • SQL_TRUE — 游标是只读的，并且不能用于定位型更新或删除。除非已将 SQL_ATTR_FOR_FETCH_ONLY 环境设置为 SQL_FALSE，否则这是缺省值。 • SQL_FALSE — 游标可用于定位型更新和删除。
SQL_ATTR_FULL_OPEN	32 位的整数值，这个值指定为此语句句柄打开的游标是否应该完全打开。 • SQL_FALSE — 为此语句句柄打开游标时，可使用高速缓存的游标来提高性能。这是缺省值。 • SQL_TRUE — 为此语句句柄打开游标时，始终强制完全打开新游标。
SQL_ATTR_ROWSET_SIZE	32 位的整数值，这个值指定行集中的行数。这是每个 SQLExtendedFetch() 调用所返回的行数。缺省值是 1。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 163. SQLStmtAttr SQLSTATE

SQLSTATE	描述	说明
40003 *****	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
HY000	一般错误	发生错误，该错误没有特定的 SQLSTATE，并且没有为其定义由实现定义的 SQLSTATE。SQLError 在自变量 *szErrorMsg* 中返回的错误消息描述了此错误及其原因。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	给定所指定的 *fAttr* 值，对自变量 vParam 指定了无效的值。 指定了无效的 *fAttr* 值。 自变量 *vParam* 是空指针。
HY010	函数顺序错误	调用此函数的顺序不正确。
HYC00	驱动程序不具有能力	驱动程序或数据源不支持指定的选项。

SQLSetStmtOption — 设置语句选项

用途

SQLSetStmtOption() 设置特定语句句柄的属性。要为与某个连接句柄相关联的所有语句句柄设置选项，应用程序可以调用 SQLSetConnectOption()（有关其它详细信息，另请参考第 210 页的『SQLSetConnectOption — 设置连接选项』）。

语法

```
SQLRETURN SQLSetStmtOption  (SQLHSTMT       hstmt,
                             SQLSMALLINT    fOption,
                             SQLPOINTER     vParam);
```

函数自变量

表 164. SQLSetStmtOption 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*fOption*	输入	要设置的选项。有关可设置的语句选项的列表，请参考第 223 页的表 162。
SQLPOINTER	*vParam*	输入	与 *fOption* 相关联的值。*vParam* 可以是指向 32 位整数值的指针，也可以是字符串。

用法

SQLSetStmtOption() 与 SQLSetStmtAttr() 提供相同的功能，支持这两个函数的目的都是为了提高兼容性。

在另一个 SQLSetStmtOption() 调用更改 *hstmt* 的语句选项或通过使用 SQL_DROP 选项调用 SQLFreeStmt() 来删除该 *hstmt* 之前，那些语句选项将保持有效。使用 SQL_CLOSE、SQL_UNBIND 或 SQL_RESET_PARAMS 选项调用 SQLFreeStmt() 并不会重设语句选项。

通过 *vParam* 设置的信息的格式取决于所指定的 *fOption*。第 223 页的表 162 说明了每种信息的格式。

请参考第 223 页的表 162 以了解正确的语句选项。

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 165. SQLStmtOption SQLSTATE

SQLSTATE	描述	说明
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。

表 165. SQLStmtOption SQLSTATE (续)

SQLSTATE	描述	说明
HY000	一般错误	发生错误，该错误没有特定的 SQLSTATE，并且没有为其定义由实现定义的 SQLSTATE。SQLError 在自变量 *szErrorMsg* 中返回的错误消息描述了此错误及其原因。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量值无效	给定所指定的 *fOption* 值，对自变量 vParam 指定了无效的值。 指定了无效的 *fOption* 值。 自变量 *szSchemaName* 或 *szTableName* 是空指针。
HY010	函数顺序错误	调用此函数的顺序不正确。
HYC00	驱动程序不具有能力	驱动程序或数据源不支持指定的选项。

SQLSpecialColumns — 获取特殊（行标识符）列

用途

SQLSpecialColumns() 返回表的唯一行标识符信息（主键或唯一索引）。例如，唯一索引或主键信息。将在 SQL 结果集中返回此信息，可以使用那些用来取装由 SELECT 语句生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLSpecialColumns  (SQLHSTMT       hstmt,
                              SQLSMALLINT    fColType,
                              SQLCHAR        *szCatalogName,
                              SQLSMALLINT    cbCatalogName,
                              SQLCHAR        *szSchemaName,
                              SQLSMALLINT    cbSchemaName,
                              SQLCHAR        *szTableName,
                              SQLSMALLINT    cbTableName,
                              SQLSMALLINT    fScope,
                              SQLSMALLINT    fNullable);
```

函数自变量

表 166. SQLSpecialColumns 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLSMALLINT	*fColType*	输入	为将来使用而保留，用于支持其它类型的特殊列。 当前，忽略此数据类型。
SQLCHAR *	*szCatalogName*	输入	由三部分组成的表名的目录限定符。这必须是空指针或长度为零的字符串。
SQLSMALLINT	*cbCatalogName*	输入	*szCatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	*szSchemaName*	输入	指定的表的模式限定符。
SQLSMALLINT	*cbSchemaName*	输入	*szSchemaName* 的长度。
SQLCHAR *	*szTableName*	输入	表名
SQLSMALLINT	*cbTableName*	输入	*cbTableName* 的长度。

表 166. SQLSpecialColumns 自变量 (续)

数据类型	自变量	使用	描述
SQLSMALLINT	*fScope*	输入	唯一行标识符必须保持有效的最短持续时间。 *fScope* 必须是下列其中一项: • SQL_SCOPE_CURROW — 仅当定位在该行上时才保证行标识符有效。如果另一个事务更新或删除了某一行，则以后使用同一个行标识符值进行的重新查询可能不会返回该行。 • SQL_SCOPE_TRANSACTION — 保证行标识符在当前事务的持续时间内有效。 • SQL_SCOPE_SESSION — 保证行标识符在连接的持续时间内有效。 保证行标识符值有效的持续时间取决于当前事务隔离级别。有关涉及隔离级别的信息和方案，请参考 IBM DB2 SQL Reference。
SQLSMALLINT	*fNullable*	输入	确定是否返回可以具有空值的特殊列。 必须是下列其中一项: • SQL_NO_NULLS 返回的行标识符列集不能具有任何空值。 • SQL_NULLABLE 返回的行标识符列集可以包括允许空值的列。

用法

如果存在多种方法来唯一地标识表中的任何行（例如，在指定的表上存在多个唯一索引），则 DB2 UDB CLI 根据其内部标准来返回行标识符列的*最佳*集合。

如果没有任何列集允许唯一地标识表中的任何行，则将返回空结果集。

唯一行标识符信息以结果集的形式返回，其中，行标识符的每一列都由结果集中的一行表示。`SQLSpecialColumns()` 返回的结果集按以下次序包含下列各列:

表 167. SQLSpecialColumns 返回的列

列名	数据类型	描述
SCOPE	SMALLINT 非空	行标识的实际作用域。包含下列其中一个值: • SQL_SCOPE_CURROW • SQL_SCOPE_TRANSACTION • SQL_SCOPE_SESSION 有关每个值的描述，请参考第 228 页的表 166 中的 *fScope*。
COLUMN_NAME	VARCHAR(128) 非空	行标识符列的名称。
DATA_TYPE	SMALLINT 非空	列的 SQL 数据类型。

表 167. SQLSpecialColumns 返回的列 (续)

列名	数据类型	描述
TYPE_NAME	VARCHAR(128) 非空	与 DATA_TYPE 列值相关联的名称的 DBMS 字符串表示法。
LENGTH_PRECISION	INTEGER	列的精度。对于精度不适用的数据类型，将返回空。
BUFFER_LENGTH	INTEGER	以缺省 C 类型返回的数据的长度（字节）。对于 CHAR 数据类型，这与 LENGTH_PRECISION 列中的值相同。
SCALE	SMALLINT	列的标度。对于标度不适用的数据类型，将返回空。
PSEUDO_COLUMN	SMALLINT	指示该列是否是伪列；DB2 UDB CLI 只返回: • SQL_PC_NOT_PSEUDO

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 168. SQLSpecialColumns SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	请求了与游标相关的信息，但没有打开游标。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量长度无效	其中一个长度自变量的值小于 0，但不等于 SQL_NTS。
HYC00	驱动程序不具有能力	数据源不支持由三部分组成的表名的*目录*部分（第一部分）。

SQLStatistics — 获取基本表的索引和统计信息

用途

SQLStatistics() 检索给定的表的索引信息。它还返回与该表相关联的基数和页数以及基于该表的索引。将在一个结果集中返回此信息，可以使用那些用来取装由 SELECT 语句生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLStatistics  (SQLHSTMT       hstmt,
                          SQLCHAR        *szCatalogName,
                          SQLSMALLINT    cbCatalogName,
                          SQLCHAR        *szSchemaName,
                          SQLSMALLINT    cbSchemaName,
                          SQLCHAR        *szTableName,
                          SQLSMALLINT    cbTableName,
                          SQLSMALLINT    fUnique,
                          SQLSMALLINT    fAccuracy);
```

函数自变量

表 169. SQLStatistics 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLCHAR *	*szCatalogName*	输入	由三部分组成的表名的目录限定符。这必须是空指针或长度为零的字符串。
SQLSMALLINT	*cbCatalogName*	输入	*cbCatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	*szSchemaName*	输入	指定的表的模式限定符。
SQLSMALLINT	*cbSchemaName*	输入	*szSchemaName* 的长度。
SQLCHAR *	*szTableName*	输入	表名
SQLSMALLINT	*cbTableName*	输入	*cbTableName* 的长度。
SQLSMALLINT	*fUnique*	输入	要返回的索引信息的类型: • SQL_INDEX_UNIQUE 只返回唯一索引。 • SQL_INDEX_ALL 返回所有索引。
SQLSMALLINT	*fAccuracy*	输入	当前不使用此自变量，必须将其设置为 0。

用法

SQLStatistics() 返回下列类型的信息:

- 表的统计信息（如果可用的话）:
  - 当下表中的 TYPE 列设置为 SQL_TABLE_STAT 时，将返回表中的行数以及用来存储该表的页数。
  - 当 TYPE 列指示索引时，返回索引中的唯一值的数目以及用来存储索引的页数。

SQLStatistics

– 关于每个索引的信息，其中，每个索引列都由结果集的一行表示。结果集列是按照下表中显示的次序给出的；结果集中的行按照 NON_UNIQUE、TYPE、INDEX_QUALIFIER、INDEX_QUALIFIER、INDEX_NAME 和 ORDINAL_POSITION 排序。

表 170. SQLStatistics 返回的列

列名	数据类型	描述
TABLE_CAT	VARCHAR(128)	包含 TABLE_SCHEM 的目录的名称。此列设置为空。
TABLE_SCHEM	VARCHAR(128)	包含 TABLE_NAME 的模式的名称。
TABLE_NAME	VARCHAR(128) 非空	表的名称。
NON_UNIQUE	SMALLINT	指示索引是否允许重复值: • 如果索引允许重复值，则为 TRUE。 • 如果索引值必须是唯一的，则为 FALSE。 • 如果 TYPE 列指示此行为 SQL_TABLE_STAT（有关表本身的统计信息），则返回空。
INDEX_QUALIFIER	VARCHAR(128)	用来限定索引名的标识符。如果 TYPE 列指示 SQL_TABLE_STAT，则此列为空。
INDEX_NAME	VARCHAR(128)	索引的名称。如果 TYPE 列的值为 SQL_TABLE_STAT，则此列的值为空。
TYPE	SMALLINT 非空	指示包含在结果集的此行中的信息的类型: • SQL_TABLE_STAT 指示此行包含有关表本身的统计信息。 • SQL_INDEX_CLUSTERED 指示此行包含有关索引的信息，并且索引类型是群集索引。 • SQL_INDEX_HASHED 指示此行包含有关索引的信息，并且索引类型是散列索引。 • SQL_INDEX_OTHER 指示此行包含有关索引的信息，并且索引类型是除群集索引和散列索引之外的索引。 **注**: 当前，SQL_INDEX_OTHER 是唯一可能的类型。
ORDINAL_POSITION	SMALLINT	列在 INDEX_NAME 列中给出了其名称的索引中的序数位置。如果 TYPE 列的值为 SQL_TABLE_STAT，则对此列返回空值。
COLUMN_NAME	VARCHAR(128)	列在索引中的名称。
COLLATION	CHAR(1)	列的排序顺序; “**A**” 表示升序, “**D**” 表示降序。如果 TYPE 列中的值是 SQL_TABLE_STAT，则将返回空值。
CARDINALITY	INTEGER	• 如果 TYPE 列包含 SQL_TABLE_STAT 值，则此列包含表中的行数。 • 如果 TYPE 列的值不是 SQL_TABLE_STAT，则此列包含索引中的唯一值的数目。 • 如果 DBMS 没有提供信息，则返回空值。
PAGES	INTEGER	• 如果 TYPE 列包含 SQL_TABLE_STAT 值，则此列包含用来存储该表的页数。 • 如果 TYPE 列的值不是 SQL_TABLE_STAT，则此列包含用来存储索引的页数。 • 如果 DBMS 没有提供信息，则返回空值。

对于结果集中的包含表统计信息的行（TYPE 设置为 SQL_TABLE_STAT），NON_UNIQUE、INDEX_QUALIFIER、INDEX_NAME、ORDINAL_POSITION、COLUMN_NAME 和 COLLATION 的列值将设置为空。如果不能确定 CARDINALITY 或 PAGES 信息，则对那些列返回空。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 171. SQLStatistics SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	请求了与游标相关的信息，但没有打开游标。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HYC00	驱动程序不具有能力	数据源不支持由三部分组成的表名的目录部分（第一部分）。

SQLTablePrivileges — 获取与表相关联的特权

用途

SQLTablePrivileges() 返回一些表以及每个表的相关联特权的列表。将在 SQL 结果集中返回此信息，可以使用那些用来处理由查询生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLTablePrivileges  (SQLHSTMT         StatementHandle,
                               SQLCHAR          *CatalogName,
                               SQLSMALLINT      NameLength1,
                               SQLCHAR          *SchemaName,
                               SQLSMALLINT      NameLength2,
                               SQLCHAR          *TableName,
                               SQLSMALLINT      NameLength3);
```

函数自变量

表 172. SQLTablePrivileges 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLCHAR *	*szTableQualifier*	输入	由三部分组成的表名的目录限定符。这必须是空指针或长度为零的字符串。
SQLSMALLINT	*cbTableQualifier*	输入	*CatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	*SchemaName*	输入	缓冲区，它可能包含*模式值*，以通过模式名限定结果集。
SQLSMALLINT	*NameLength2*	输入	*SchemaName* 的长度。
SQLCHAR *	*TableName*	输入	缓冲区，它可能包含*模式值*，以通过表名限定结果集。
SQLSMALLINT	*NameLength3*	输入	*TableName* 的长度。

用法

结果是作为包含下表中列示的列的标准结果集返回的。此结果集按 TABLE_CAT、TABLE_SCHEM、TABLE_NAME 和 PRIVILEGE 进行排序。如果有多个特权与任何给定的表相关联，则将每个特权作为单独的行返回。

此处报告的每个特权的粒度可能可以也可能不可以在列级别应用；例如，对于某些数据源，如果可以更新表，则也可以更新该表中的每一列。对于其它数据源，应用程序必须调用 SQLColumnPrivileges() 来查看个别的列是否具有相同的表特权。

由于在许多情况下调用 SQLColumnPrivileges() 都意味着要对系统目录进行复杂的查询（这样的成本很高），所以，您应该有节制地使用这些调用，并且应该保存结果而不是重复地进行调用。

为了与 SQL92 限制一致，编目函数结果集的 VARCHAR 列是使用最大长度属性 128 声明的。由于 DB2 名称的长度小于 128，所以应用程序可以选择始终为输出缓冲区留出 128 个字符（加上空终止符）的空间，也可以选择使用 SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN 和 SQL_MAX_COLUMN_NAME_LEN 来调用 SQLGetInfo() 以分别确定所连接的 DBMS 支持的 TABLE_CAT、TABLE_SCHEM、TABLE_NAME 和 COLUMN_NAME 列的实际长度。

虽然在将来的发行版中可能会添加新列和更改现有列的名称，但当前列的位置不会更改。

表 173. SQLTablePrivileges 返回的列

列名	数据类型	描述
TABLE_CAT	VARCHAR(128)	此列始终为空。
TABLE_SCHEM	VARCHAR(128)	包含 TABLE_NAME 的模式的名称。
TABLE_NAME	VARCHAR(128) 非空	表的名称。
GRANTOR	VARCHAR(128)	进行特权授权的用户的授权标识。
GRANTEE	VARCHAR(128)	接受特权授权的用户的授权标识。
PRIVILEGE	VARCHAR(128)	表特权。这可以是下列其中一个字符串: • ALTER • CONTROL • INDEX • DELETE • INSERT • REFERENCES • SELECT • UPDATE
IS_GRANTABLE	VARCHAR(3)	指示是否允许被授权人将特权授予其它用户。 这可以是 “YES”、 “NO” 或 “NULL”。

注意: DB2 CLI 使用的列名遵循 X/Open CLI CAE 规范样式。列类型、内容和次序与 ODBC 中对 SQLProcedures() 结果集定义的那些列类型、内容和次序完全相同。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 174. SQLTablePrivileges SQLSTATE

SQLSTATE	描述	说明
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	字符串或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HY010	函数顺序错误	语句句柄的游标已打开。 没有用于此语句句柄的连接。

限制

无。

示例

```
/* From the CLI sample TBINFO.C */
/* ... */

    /* call SQLTablePrivileges */
    printf("\n    Call SQLTablePrivileges for:\n");
    printf("        tbSchemaPattern = %s\n", tbSchemaPattern);
    printf("        tbNamePattern = %s\n", tbNamePattern);
    sqlrc = SQLTablePrivileges( hstmt, NULL, 0,
                                tbSchemaPattern, SQL_NTS,
                                tbNamePattern, SQL_NTS);
    STMT_HANDLE_CHECK( hstmt, sqlrc);
```

参考

SQLTables — 获取表信息

用途

SQLTables() 返回表名和相关联信息的列表，该信息存储在所连接的数据源的系统目录中。表名列表是作为结果集返回的，可以使用那些用来检索由 SELECT 语句生成的结果集的函数来检索此结果集。

语法

```
SQLRETURN SQLTables  (SQLHSTMT       hstmt,
                      SQLCHAR        *szCatalogName,
                      SQLSMALLINT    cbCatalogName,
                      SQLCHAR        *szSchemaName,
                      SQLSMALLINT    cbSchemaName,
                      SQLCHAR        *szTableName,
                      SQLSMALLINT    cbTableName,
                      SQLCHAR        *szTableType,
                      SQLSMALLINT    cbTableType);
```

函数自变量

表 175. SQLTables 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*hstmt*	输入	语句句柄
SQLCHAR *	*szCatalogName*	输入	缓冲区，它可能包含*模式值*，以限定结果集。*目录*是由三部分组成的表名的第一部分。 这必须是空指针或长度为零的字符串。
SQLSMALLINT	*cbCatalogName*	输入	*szCatalogName* 的长度。必须将此长度设置为 0。
SQLCHAR *	*szSchemaName*	输入	缓冲区，它可能包含*模式值*，以通过模式名限定结果集。
SQLSMALLINT	*cbSchemaName*	输入	*szSchemaName* 的长度。
SQLCHAR *	*szTableName*	输入	缓冲区，它可能包含*模式值*，以通过表名限定结果集。
SQLSMALLINT	*cbTableName*	输入	*szTableName* 的长度。
SQLCHAR *	*szTableType*	输入	缓冲区，它可能包含*值列表*，以通过表类型限定结果集。 值列表是您感兴趣的类型的值的列表，各个值之间用逗号分隔。有效表类型标识符可以包括：ALL、BASE TABLE、TABLE、VIEW 和 SYSTEM TABLE。如果 *szTableType* 自变量是空指针或长度为零的字符串，则这等同于对表类型标识符指定所有可能的值。 如果指定 SYSTEM TABLE，则将同时返回系统表和系统视图（如果有的话）。 指定表类型时，可以使用引号，也可以不使用引号。
SQLSMALLINT	*cbTableType*	输入	*szTableType* 的大小

SQLTables

注意，*szCatalogName*、*szSchemaName* 和 *szTableName* 自变量接受搜索模式。

可以随通配符一起指定转义字符，以便可以在搜索模式中使用实际字符。转义字符是在 SQL_ATTR_ESCAPE_CHAR 环境属性中指定的。

用法

表信息在一个结果集中返回，其中，每个表都由结果集的一行表示。

SQLTables() 返回的结果集包含下表中列示的列，并且具有所给出的次序。

表 176. SQLTables 返回的列

列名	数据类型	描述
TABLE_CAT	VARCHAR(128)	当前服务器。
TABLE_SCHEM	VARCHAR(128)	包含 TABLE_NAME 的模式的名称。
TABLE_NAME	VARCHAR(128)	表、视图、别名或同义词的名称。
TABLE_TYPE	VARCHAR(128)	标识由 TABLE_NAME 列中的名称给出的类型。它可以具有字符串值 “TABLE”、“VIEW”、“BASE TABLE” 或 “SYSTEM TABLE”。
REMARKS	VARCHAR(254)	包含关于表的描述性信息。

返回码

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 177. SQLTables SQLSTATE

SQLSTATE	描述	说明
24000	游标状态无效	请求了与游标相关的信息，但没有打开游标。
40003 *	未知语句是否完成	在函数完成处理之前，CLI 与数据源之间的通信链路发生故障。
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY009	自变量或缓冲区长度无效	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。
HYC00	驱动程序不具有能力	数据源不支持由三部分组成的表名的目录部分（第一部分）。

SQLTransact — 事务管理

用途

SQLTransact() 落实或回滚连接中的当前事务。

将落实或回滚从进行连接时开始或从上次调用 SQLTransact() 时开始（以较后者为准）在该连接上对数据库执行的所有更改。

如果某个事务正在该连接上活动，则应用程序在可以与数据库断开连接之前必须调用 SQLTransact()。

语法

```
SQLRETURN SQLTransact (SQLHENV        henv,
                       SQLHDBC        hdbc,
                       SQLSMALLINT    fType);
```

函数自变量

表 178. SQLTransact 自变量

数据类型	自变量	使用	描述
SQLHENV	*henv*	输入	环境句柄。 如果 *hdbc* 是有效的连接句柄，则忽略 *henv*。
SQLHDBC	*hdbc*	输入	数据库连接句柄。 如果将 *hdbc* 设置为 SQL_NULL_HDBC，则 *henv* 必须包含与连接相关联的环境句柄。
SQLSMALLINT	*fType*	输入	期望的事务操作。此自变量必须具有下列其中一个值: • SQL_COMMIT • SQL_ROLLBACK • SQL_COMMIT_HOLD • SQL_ROLLBACK_HOLD

用法

使用 SQL_COMMIT 或 SQL_ROLLBACK 完成事务具有下列效果:

- 在调用 SQLTransact() 之后，语句句柄仍有效。
- 游标名、绑定的参数和列绑定可以跨事务而存在。
- 关闭已打开的游标，并且废弃任何处于暂挂检索状态的结果集。

使用 SQL_COMMIT_HOLD 或 SQL_ROLLBACK_HOLD 来完成事务仍将落实或回滚数据库更改，但不会导致关闭游标。

如果当前在连接上没有活动事务，则调用 SQLTransact() 对数据库服务器没有影响，并将返回 SQL_SUCCESS。

在执行 COMMIT 或 ROLLBACK 时，SQLTransact() 可能会因为丢失连接而失败。在这种情况下，应用程序可能无法确定是否已处理了 COMMIT 或 ROLLBACK，并且可能需要数据库管理员的帮助。有关事务记录和其它事务管理任务的更多信息，请参考 DBMS 产品信息。

SQLTransact

返回码

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断

表 179. SQLTransact SQLSTATE

SQLSTATE	描述	说明
08003	连接未打开	*hdbc* 未处于已连接状态。
08007	在事务进行期间，连接发生故障。	在函数执行期间，与 *hdbc* 相关联的连接发生故障，并且无法确定在发生故障之前是否已执行所请求的 COMMIT 或 ROLLBACK。
58004	系统错误	不可恢复的系统错误
HY001	内存分配失败	驱动程序无法分配支持此函数的执行或完成所必需的内存。
HY012	事务操作状态无效	对自变量 *fType* 指定的值既不是 SQL_COMMIT 也不是 SQL_ROLLBACK。
HY013 *	内存管理问题	驱动程序无法访问支持此函数的执行或完成所必需的内存。

示例

请参考第 98 页的『示例』。

附录 A. DB2 UDB CLI 一般诊断信息

本附录部分包含本书各章节引用的信息的表。

DB2 UDB CLI 函数返回码

返回码	值	描述
SQL_SUCCESS	0	函数成功完成，没有附加的 SQLSTATE 信息可用。
SQL_SUCCESS_WITH_INFO	1	函数成功完成，但有警告或其它信息。请调用 SQLError() 以接收 SQLSTATE 和其它错误信息。
SQL_NO_DATA_FOUND	100	函数成功返回，但找不到相关的信息。
SQL_ERROR	-1	函数失败。请调用 SQLError() 以接收 SQLSTATE 和任何其它错误信息。
SQL_INVALID_HANDLE	-2	由于作为输入自变量传送的句柄（环境、连接或语句句柄）无效，所以函数失败。

附录 B. DB2 UDB CLI 包含文件

DB2 UDB CLI 中使用的唯一一个包含文件就是 sqlcli.h。

```
/*** START HEADER FILE SPECIFICATIONS ******************************/
/*                                                                  */
/* Header File Name: SQLCLI                                         */
/*                                                                  */
/* Descriptive Name: Structured Query Language (SQL) Call Level     */
/*                   Interface.                                     */
/*                                                                  */
/* 5716-SS1  (C) Copyright IBM Corp. 1995,1995                      */
/* All rights reserved.                                             */
/* US Government Users Restricted Rights -                          */
/* Use, duplication or disclosure restricted                        */
/* by GSA ADP Schedule Contract with IBM Corp.                      */
/*                                                                  */
/* Licensed Materials-Property of IBM                               */
/*                                                                  */
/*                                                                  */
/* Description: The SQL Call Level Interface provides access to     */
/*              most SQL functions, without the need for a          */
/*              precompiler.                                        */
/*                                                                  */
/* Header Files Included: SQLCLI                                    */
/*                                                                  */
/* Function Prototype List:  SQLAllocConnect                        */
/*                           SQLAllocEnv                            */
/*                           SQLAllocHandle                         */
/*                           SQLAllocStmt                           */
/*                           SQLBindCol                             */
/*                           SQLBindFileToCol                       */
/*                           SQLBindFileToParam                     */
/*                           SQLBindParam                           */
/*                           SQLBindParameter                       */
/*                           SQLCancel                              */
/*                           SQLCloseCursor                         */
/*                           SQLColAttributes                       */
/*                           SQLColumns                             */
/*                           SQLConnect                             */
/*                           SQLCopyDesc                            */
/*                           SQLDataSources                         */
/*                           SQLDescribeCol                         */
/*                           SQLDescribeParam                       */
/*                           SQLDisconnect                          */
/*                           SQLDriverConnect                       */
/*                           SQLEndTran                             */
/*                           SQLError                               */
/*                           SQLExecDirect                          */
/*                           SQLExecute                             */
/*                           SQLExtendedFetch                       */
/*                           SQLFetch                               */
/*                           SQLFetchScroll                         */
/*                           SQLForeignKeys                         */
/*                           SQLFreeConnect                         */
/*                           SQLFreeEnv                             */
/*                           SQLFreeHandle                          */
/*                           SQLFreeStmt                            */
/*                           SQLGetCol                              */
/*                           SQLGetConnectOption                    */
/*                           SQLGetCursorName                       */
/*                           SQLGetConnectAttr                      */
/*                           SQLGetData                             */
/*                           SQLGetDescField                        */
/*                           SQLGetDescRec                          */
```

```
 /*                          SQLGetDiagField                     */
 /*                          SQLGetDiagRec                       */
 /*                          SQLGetEnvAttr                       */
 /*                          SQLGetFunctions                     */
 /*                          SQLGetInfo                          */
 /*                          SQLGetLength                        */
 /*                          SQLGetPosition                      */
 /*                          SQLGetStmtAttr                      */
 /*                          SQLGetStmtOption                    */
 /*                          SQLGetSubString                     */
 /*                          SQLGetTypeInfo                      */
 /*                          SQLLanguages                        */
 /*                          SQLMoreResults                      */
 /*                          SQLNativeSql                        */
 /*                          SQLNumParams                        */
 /*                          SQLNumResultCols                    */
 /*                          SQLParamData                        */
 /*                          SQLParamOptions                     */
 /*                          SQLPrepare                          */
 /*                          SQLPrimaryKeys                      */
 /*                          SQLProcedureColumns                 */
 /*                          SQLProcedures                       */
 /*                          SQLPutData                          */
 /*                          SQLReleaseEnv                       */
 /*                          SQLRowCount                         */
 /*                          SQLSetConnectAttr                   */
 /*                          SQLSetConnectOption                 */
 /*                          SQLSetCursorName                    */
 /*                          SQLSetDescField                     */
 /*                          SQLSetDescRec                       */
 /*                          SQLSetEnvAttr                       */
 /*                          SQLSetParam                         */
 /*                          SQLSetStmtAttr                      */
 /*                          SQLSetStmtOption                    */
 /*                          SQLSpecialColumns                   */
 /*                          SQLStatistics                       */
 /*                          SQLTables                           */
 /*                          SQLTransact                         */
 /*                                                              */
 /* Change Activity:                                             */
 /*                                                              */
 /* CFD List:                                                    */
 /*                                                              */
 /* FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION  */
 /* ---- ------------ ----- ------ --------- --------------------*/
 /* $A0= D91823       3D60  941206 MEGERIAN  New Include         */
 /* $A1= D94881       4D20  960816 MEGERIAN  V4R2M0 enhancements */
 /* $A2= D95600       4D30  970910 MEGERIAN  V4R3M0 enhancements */
 /* $A3= P3682850     4D40  981030 MEGERIAN  V4R4M0 enhancements */
 /* $A4= D97596       4D50  990326 LJAMESON  V4R5M0 enhancements */
 /*                                                              */
 /* End CFD List.                                                */
 /*                                                              */
 /*  Additional notes about the Change Activity                  */
 /* End Change Activity.                                         */
 /*** END HEADER FILE SPECIFICATIONS ******************************/

#ifndef SQL_H_SQLCLI
   #define SQL_H_SQLCLI            /* Permit duplicate Includes */

#ifndef __SQL_EXTERN
    #ifdef __ILEC400__
        #define SQL_EXTERN extern
    #else
        #ifdef __cplusplus
            #define SQL_EXTERN extern "C nowiden"
        #else
```

```
            #define SQL_EXTERN extern "C"
        #endif
    #endif
    #define __SQL_EXTERN
#endif

/* generally useful constants */
#define  SQL_FALSE                 0
#define  SQL_TRUE                  1
#define  SQL_NTS                  -3  /* NTS = Null Terminated String    */
#define  SQL_SQLSTATE_SIZE         5  /* size of SQLSTATE, not including
                                         null terminating byte           */
#define  SQL_MAX_MESSAGE_LENGTH    512

/* RETCODE values             */
#define  SQL_SUCCESS              0
#define  SQL_SUCCESS_WITH_INFO    1
#define  SQL_NO_DATA_FOUND        100
#define  SQL_NEED_DATA            99
#define  SQL_NO_DATA              SQL_NO_DATA_FOUND
#define  SQL_ERROR               -1
#define  SQL_INVALID_HANDLE      -2

/* SQLFreeStmt option values  */
#define  SQL_CLOSE                0
#define  SQL_DROP                 1
#define  SQL_UNBIND               2
#define  SQL_RESET_PARAMS         3

/* SQLSetParam defines        */
#define  SQL_C_DEFAULT           99

/* SQLTransact option values  */
#define  SQL_COMMIT               0
#define  SQL_ROLLBACK             1
#define  SQL_COMMIT_HOLD          2
#define  SQL_ROLLBACK_HOLD        3

/* SQLDriverConnect option values  */
#define  SQL_DRIVER_COMPLETE          1
#define  SQL_DRIVER_COMPLETE_REQUIRED 1
#define  SQL_DRIVER_NOPROMPT          1

/*  Valid option codes for GetInfo procedure               */
#define  SQL_ACTIVE_CONNECTIONS    0
#define  SQL_ACTIVE_STATEMENTS     1
#define  SQL_PROCEDURES            2
#define  SQL_DBMS_NAME             17
#define  SQL_DBMS_VER              18
#define  SQL_MAX_COLUMN_NAME_LEN   30
#define  SQL_MAX_CURSOR_NAME_LEN   31
#define  SQL_MAX_OWNER_NAME_LEN    32
#define  SQL_MAX_SCHEMA_NAME_LEN   33
#define  SQL_MAX_TABLE_NAME_LEN    35

/* Standard SQL data types */
#define  SQL_CHAR                  1
#define  SQL_NUMERIC               2
#define  SQL_DECIMAL               3
#define  SQL_INTEGER               4
#define  SQL_SMALLINT              5
#define  SQL_FLOAT                 6
#define  SQL_REAL                  7
#define  SQL_DOUBLE                8
#define  SQL_DATETIME              9
#define  SQL_VARCHAR              12
#define  SQL_BLOB                 13
```

```
#define  SQL_CLOB               14
#define  SQL_DBCLOB             15
#define  SQL_DATALINK           16
#define  SQL_WCHAR              17
#define  SQL_WVARCHAR           18
#define  SQL_BIGINT             19
#define  SQL_BLOB_LOCATOR       20
#define  SQL_CLOB_LOCATOR       21
#define  SQL_DBCLOB_LOCATOR     22
#define  SQL_WLONGVARCHAR       SQL_WVARCHAR
#define  SQL_LONGVARCHAR        SQL_VARCHAR
#define  SQL_GRAPHIC            95
#define  SQL_VARGRAPHIC         96
#define  SQL_LONGVARGRAPHIC     SQL_VARGRAPHIC
#define  SQL_BINARY             97
#define  SQL_VARBINARY          98
#define  SQL_LONGVARBINARY      SQL_VARBINARY
#define  SQL_DATE               91
#define  SQL_TYPE_DATE          91
#define  SQL_TIME               92
#define  SQL_TYPE_TIME          92
#define  SQL_TIMESTAMP          93
#define  SQL_TYPE_TIMESTAMP     93
#define  SQL_CODE_DATE           1
#define  SQL_CODE_TIME           2
#define  SQL_CODE_TIMESTAMP      3
#define  SQL_ALL_TYPES           0

/*
 * NULL status defines; these are used in SQLColAttributes, SQLDescribeCol,
 * to describe the nullability of a column in a table.
 */
#define  SQL_UNUSED                          0
#define  SQL_HANDLE_ENV                      1
#define  SQL_HANDLE_DBC                      2
#define  SQL_HANDLE_STMT                     3
#define  SQL_HANDLE_DESC                     4
#define  SQL_NULL_HANDLE                     0

#define  SQL_NO_NULLS         0
#define  SQL_NULLABLE         1
#define  SQL_NULLABLE_UNKNOWN 2

/* Special length values  */
#define  SQL_NULL_DATA       -1
#define  SQL_DATA_AT_EXEC    -2
#define  SQL_BIGINT_PREC     19
#define  SQL_INTEGER_PREC    10
#define  SQL_SMALLINT_PREC    5


/* SQLColAttributes defines */
#define  SQL_ATTR_READONLY           0
#define  SQL_ATTR_WRITE              1
#define  SQL_ATTR_READWRITE_UNKNOWN  2

/* Valid concurrency values */
#define  SQL_CONCUR_LOCK             0
#define  SQL_CONCUR_READ_ONLY        1

/*  Valid environment attributes                    */
#define SQL_ATTR_OUTPUT_NTS          10001
#define SQL_ATTR_SYS_NAMING          10002
#define SQL_ATTR_DEFAULT_LIB         10003
#define SQL_ATTR_SERVER_MODE         10004
#define SQL_ATTR_JOB_SORT_SEQUENCE   10005
#define SQL_ATTR_ENVHNDL_COUNTER     10009
```

```
#define SQL_ATTR_ESCAPE_CHAR         10010

/*  Valid environment/connection attributes           */
#define SQL_ATTR_DATE_FMT            10020
#define SQL_ATTR_DATE_SEP            10021
#define SQL_ATTR_TIME_FMT            10022
#define SQL_ATTR_TIME_SEP            10023
#define SQL_ATTR_DECIMAL_SEP         10024

/*  Valid environment/connection values               */
#define SQL_FMT_ISO                  1
#define SQL_FMT_USA                  2
#define SQL_FMT_EUR                  3
#define SQL_FMT_JIS                  4
#define SQL_FMT_MDY                  5
#define SQL_FMT_DMY                  6
#define SQL_FMT_YMD                  7
#define SQL_FMT_JUL                  8
#define SQL_FMT_HMS                  9
#define SQL_FMT_JOB                  10
#define SQL_SEP_SLASH                1
#define SQL_SEP_DASH                 2
#define SQL_SEP_PERIOD               3
#define SQL_SEP_COMMA                4
#define SQL_SEP_BLANK                5
#define SQL_SEP_COLON                6
#define SQL_SEP_JOB                  7

/*  Valid values for type in GetCol                                 */
#define SQL_DEFAULT              99
#define SQL_ARD_TYPE            -99

/*  Valid values for UPDATE_RULE and DELETE_RULE in SQLForeignKeys  */
#define SQL_CASCADE              1
#define SQL_RESTRICT             2
#define SQL_NO_ACTION            3
#define SQL_SET_NULL             4
#define SQL_SET_DEFAULT          5

/*  Valid values for COLUMN_TYPE in SQLProcedureColumns  */
#define SQL_PARAM_INPUT             1
#define SQL_PARAM_OUTPUT            2
#define SQL_PARAM_INPUT_OUTPUT      3

/* statement attributes */
#define SQL_ATTR_APP_ROW_DESC      10010
#define SQL_ATTR_APP_PARAM_DESC    10011
#define SQL_ATTR_IMP_ROW_DESC      10012
#define SQL_ATTR_IMP_PARAM_DESC    10013
#define SQL_ATTR_FOR_FETCH_ONLY    10014
#define SQL_ATTR_CONCURRENCY       10014
#define SQL_CONCURRENCY            10014
#define SQL_ATTR_CURSOR_SCROLLABLE 10015
#define SQL_ATTR_ROWSET_SIZE       10016
#define SQL_ROWSET_SIZE            10016

 /*  Codes used in FetchScroll                                       */
#define SQL_FETCH_NEXT                   1
#define SQL_FETCH_FIRST                  2
#define SQL_FETCH_LAST                   3
#define SQL_FETCH_PRIOR                  4
#define SQL_FETCH_ABSOLUTE               5
#define SQL_FETCH_RELATIVE               6

/* SQLColAttributes defines */
#define SQL_DESC_COUNT                   1
#define SQL_DESC_TYPE                    2
```

```
#define SQL_DESC_LENGTH                 3
#define SQL_DESC_LENGTH_PTR             4
#define SQL_DESC_PRECISION              5
#define SQL_DESC_SCALE                  6
#define SQL_DESC_DATETIME_INTERVAL_CODE  7
#define SQL_DESC_NULLABLE               8
#define SQL_DESC_INDICATOR_PTR          9
#define SQL_DESC_DATA_PTR               10
#define SQL_DESC_NAME                   11
#define SQL_DESC_UNNAMED                12
#define SQL_DESC_DISPLAY_SIZE           13
#define SQL_DESC_ALLOC_TYPE             99
#define SQL_DESC_ALLOC_AUTO             1
#define SQL_DESC_ALLOC_USER             2

#define SQL_COLUMN_COUNT                  1
#define SQL_COLUMN_TYPE                   2
#define SQL_COLUMN_LENGTH                 3
#define SQL_COLUMN_LENGTH_PTR             4
#define SQL_COLUMN_PRECISION              5
#define SQL_COLUMN_SCALE                  6
#define SQL_COLUMN_DATETIME_INTERVAL_CODE  7
#define SQL_COLUMN_NULLABLE               8
#define SQL_COLUMN_INDICATOR_PTR          9
#define SQL_COLUMN_DATA_PTR               10
#define SQL_COLUMN_NAME                   11
#define SQL_COLUMN_UNNAMED                12
#define SQL_COLUMN_DISPLAY_SIZE           13
#define SQL_COLUMN_ALLOC_TYPE             99
#define SQL_COLUMN_ALLOC_AUTO             1
#define SQL_COLUMN_ALLOC_USER             2

/*  Valid codes for SpecialColumns procedure             */
#define SQL_SCOPE_CURROW         0
#define SQL_SCOPE_TRANSACTION    1
#define SQL_SCOPE_SESSION        2
#define SQL_PC_UNKNOWN           0
#define SQL_PC_NOT_PSEUDO        1
#define SQL_PC_PSEUDO            2

/*  Valid values for connect attribute                    */
#define SQL_ATTR_AUTO_IPD       10001
#define SQL_ATTR_ACCESS_MODE    10002
#define SQL_ACCESS_MODE         10002
#define SQL_ATTR_AUTOCOMMIT     10003
#define SQL_AUTOCOMMIT          10003
#define SQL_ATTR_DBC_SYS_NAMING 10004
#define SQL_ATTR_DBC_DEFAULT_LIB 10005
#define SQL_ATTR_COMMIT             0
#define SQL_MODE_READ_ONLY          0
#define SQL_MODE_READ_WRITE         1
#define SQL_MODE_DEFAULT            1
#define SQL_AUTOCOMMIT_OFF          0
#define SQL_AUTOCOMMIT_ON           1
#define SQL_TXN_ISOLATION           0
#define SQL_COMMIT_NONE             1
#define SQL_TXN_NO_COMMIT           1
#define SQL_TXN_NOCOMMIT            1
#define SQL_COMMIT_CHG              2
#define SQL_COMMIT_UR               2
#define SQL_TXN_READ_UNCOMMITTED    2
#define SQL_COMMIT_CS               3
#define SQL_TXN_READ_COMMITTED      3
#define SQL_COMMIT_ALL              4
#define SQL_COMMIT_RS               4
#define SQL_TXN_REPEATABLE_READ     4
#define SQL_COMMIT_RR               5
```

```
#define SQL_TXN_SERIALIZABLE        5

/*  Valid index flags                       */
#define SQL_INDEX_UNIQUE            0
#define SQL_INDEX_ALL               1
#define SQL_INDEX_OTHER             3

/*  Valid File Options                      */
#define SQL_FILE_READ               2
#define SQL_FILE_CREATE             8
#define SQL_FILE_OVERWRITE         16
#define SQL_FILE_APPEND            32

/* Valid types for GetDiagField                          */
#define SQL_DIAG_RETURNCODE         1
#define SQL_DIAG_NUMBER             2
#define SQL_DIAG_ROW_COUNT          3
#define SQL_DIAG_SQLSTATE           4
#define SQL_DIAG_NATIVE             5
#define SQL_DIAG_MESSAGE_TEXT       6
#define SQL_DIAG_DYNAMIC_FUNCTION   7
#define SQL_DIAG_CLASS_ORIGIN       8
#define SQL_DIAG_SUBCLASS_ORIGIN    9
#define SQL_DIAG_CONNECTION_NAME    10
#define SQL_DIAG_SERVER_NAME        11

/*
 * SQLColAttributes defines
 * These are also used by SQLGetInfo
 */
#define  SQL_UNSEARCHABLE             0
#define  SQL_LIKE_ONLY                1
#define  SQL_ALL_EXCEPT_LIKE          2
#define  SQL_SEARCHABLE               3

/* GetFunctions() values to identify CLI functions   */
#define  SQL_API_SQLALLOCCONNECT        1
#define  SQL_API_SQLALLOCENV            2
#define  SQL_API_SQLALLOCHANDLE      1001
#define  SQL_API_SQLALLOCSTMT           3
#define  SQL_API_SQLBINDCOL             4
#define  SQL_API_SQLBINDFILETOCOL    2002
#define  SQL_API_SQLBINDFILETOPARAM  2003
#define  SQL_API_SQLBINDPARAM        1002
#define  SQL_API_SQLBINDPARAMETER    1023
#define  SQL_API_SQLCANCEL              5
#define  SQL_API_SQLCLOSECURSOR      1003
#define  SQL_API_SQLCOLATTRIBUTES       6
#define  SQL_API_SQLCOLUMNS            40
#define  SQL_API_SQLCONNECT             7
#define  SQL_API_SQLCOPYDESC         1004
#define  SQL_API_SQLDATASOURCES        57
#define  SQL_API_SQLDESCRIBECOL         8
#define  SQL_API_SQLDESCRIBEPARAM      58
#define  SQL_API_SQLDISCONNECT          9
#define  SQL_API_SQLDRIVERCONNECT      68
#define  SQL_API_SQLENDTRAN          1005
#define  SQL_API_SQLERROR              10
#define  SQL_API_SQLEXECDIRECT         11
#define  SQL_API_SQLEXECUTE            12
#define  SQL_API_SQLEXTENDEDFETCH    1022
#define  SQL_API_SQLFETCH              13
#define  SQL_API_SQLFETCHSCROLL      1021
#define  SQL_API_SQLFOREIGNKEYS        60
#define  SQL_API_SQLFREECONNECT        14
#define  SQL_API_SQLFREEENV            15
#define  SQL_API_SQLFREEHANDLE       1006
```

```
#define  SQL_API_SQLFREESTMT           16
#define  SQL_API_SQLGETCOL             43
#define  SQL_API_SQLGETCONNECTATTR   1007
#define  SQL_API_SQLGETCONNECTOPTION   42
#define  SQL_API_SQLGETCURSORNAME      17
#define  SQL_API_SQLGETDATA            43
#define  SQL_API_SQLGETDESCFIELD     1008
#define  SQL_API_SQLGETDESCREC       1009
#define  SQL_API_SQLGETDIAGFIELD     1010
#define  SQL_API_SQLGETDIAGREC       1011
#define  SQL_API_SQLGETENVATTR       1012
#define  SQL_API_SQLGETFUNCTIONS       44
#define  SQL_API_SQLGETINFO            45
#define  SQL_API_SQLGETLENGTH        2004
#define  SQL_API_SQLGETPOSITION      2005
#define  SQL_API_SQLGETSTMTATTR      1014
#define  SQL_API_SQLGETSTMTOPTION      46
#define  SQL_API_SQLGETSUBSTRING     2006
#define  SQL_API_SQLGETTYPEINFO        47
#define  SQL_API_SQLLANGUAGES        2001
#define  SQL_API_SQLMORERESULTS        61
#define  SQL_API_SQLNATIVESQL          62
#define  SQL_API_SQLNUMPARAMS          63
#define  SQL_API_SQLNUMRESULTCOLS      18
#define  SQL_API_SQLPARAMDATA          48
#define  SQL_API_SQLPARAMOPTIONS     2007
#define  SQL_API_SQLPREPARE            19
#define  SQL_API_SQLPRIMARYKEYS        65
#define  SQL_API_SQLPROCEDURECOLUMNS   66
#define  SQL_API_SQLPROCEDURES         67
#define  SQL_API_SQLPUTDATA            49
#define  SQL_API_SQLRELEASEENV       1015
#define  SQL_API_SQLROWCOUNT           20
#define  SQL_API_SQLSETCONNECTATTR   1016
#define  SQL_API_SQLSETCONNECTOPTION   50
#define  SQL_API_SQLSETCURSORNAME      21
#define  SQL_API_SQLSETDESCFIELD     1017
#define  SQL_API_SQLSETDESCREC       1018
#define  SQL_API_SQLSETENVATTR       1019
#define  SQL_API_SQLSETPARAM           22
#define  SQL_API_SQLSETSTMTATTR      1020
#define  SQL_API_SQLSETSTMTOPTION      51
#define  SQL_API_SQLSPECIALCOLUMNS     52
#define  SQL_API_SQLSTATISTICS         53
#define  SQL_API_SQLTABLES             54
#define  SQL_API_SQLTRANSACT           23

/* NULL handle defines    */
#define  SQL_NULL_HENV                0L
#define  SQL_NULL_HDBC                0L
#define  SQL_NULL_HSTMT               0L

#if !defined(SDWORD)
typedef long int            SDWORD;
#endif
#if !defined(UDWORD)
typedef unsigned long int   UDWORD;
#endif
#if !defined(UWORD)
typedef unsigned short int  UWORD;
#endif
#if !defined(SWORD)
typedef signed short int    SWORD;
#endif

typedef  char              SQLCHAR;
typedef  long     int      SQLINTEGER;
```

```
typedef  short    int      SQLSMALLINT;
typedef  UWORD             SQLUSMALLINT;
typedef  UDWORD            SQLUINTEGER;
typedef  double            SQLDOUBLE;
typedef  float             SQLREAL;

typedef  void *            PTR;
typedef  PTR               SQLPOINTER;
typedef  long              HENV;
typedef  long              HDBC;
typedef  long              HSTMT;
typedef  long              HDESC;
typedef  HENV              SQLHENV;
typedef  HDBC              SQLHDBC;
typedef  HSTMT             SQLHSTMT;
typedef  HDESC             SQLHDESC;

typedef  SQLINTEGER        RETCODE;
typedef  RETCODE           SQLRETURN;

typedef  float             SFLOAT;



/*
 * DATE, TIME, and TIMESTAMP structures.  These are for compatibility
 * purposes only.  When actually specifying or retrieving DATE, TIME,
 * and TIMESTAMP values, character strings must be used.
 */


typedef  struct DATE_STRUCT
  {
    SQLSMALLINT   year;
    SQLSMALLINT   month;
    SQLSMALLINT   day;
  } DATE_STRUCT;

typedef  struct TIME_STRUCT
  {
    SQLSMALLINT   hour;
    SQLSMALLINT   minute;
    SQLSMALLINT   second;
  } TIME_STRUCT;

typedef  struct TIMESTAMP_STRUCT
  {
    SQLSMALLINT   year;
    SQLSMALLINT   month;
    SQLSMALLINT   day;
    SQLSMALLINT   hour;
    SQLSMALLINT   minute;
    SQLSMALLINT   second;
    SQLINTEGER    fraction;     /* fraction of a second */
  } TIMESTAMP_STRUCT;




SQL_EXTERN SQLRETURN  SQLAllocConnect (SQLHENV             henv,
                                       SQLHDBC            *phdbc);


SQL_EXTERN SQLRETURN  SQLAllocEnv     (SQLHENV             *phenv);
```

```
SQL_EXTERN SQLRETURN  SQLAllocHandle  (SQLSMALLINT           htype,
                                SQLINTEGER            ihnd,
                                SQLINTEGER           *ohnd);

SQL_EXTERN SQLRETURN  SQLAllocStmt    (SQLHDBC           hdbc,
                                SQLHSTMT         *phstmt);

SQL_EXTERN SQLRETURN  SQLBindCol      (SQLHSTMT          hstmt,
                                SQLSMALLINT       icol,
                                SQLSMALLINT       iType,
                                SQLPOINTER        rgbValue,
                                SQLINTEGER        cbValueMax,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindFileToCol (SQLHSTMT          hstmt,
                                SQLSMALLINT       icol,
                                SQLCHAR          *fName,
                                SQLSMALLINT      *fNameLen,
                                SQLINTEGER       *fOptions,
                                SQLSMALLINT       fValueMax,
                                SQLINTEGER       *sLen,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindFileToParam (SQLHSTMT          hstmt,
                                SQLSMALLINT       ipar,
                                SQLSMALLINT       iType,
                                SQLCHAR          *fName,
                                SQLSMALLINT      *fNameLen,
                                SQLINTEGER       *fOptions,
                                SQLSMALLINT       fValueMax,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindParam    (SQLHSTMT          hstmt,
                                SQLSMALLINT       iparm,
                                SQLSMALLINT       iType,
                                SQLSMALLINT       pType,
                                SQLINTEGER        pLen,
                                SQLSMALLINT       pScale,
                                SQLPOINTER        pData,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLBindParameter (SQLHSTMT          hstmt,
                                SQLSMALLINT       ipar,
                                SQLSMALLINT       fParamType,
                                SQLSMALLINT       fCType,
                                SQLSMALLINT       fSQLType,
                                SQLINTEGER        pLen,
                                SQLSMALLINT       pScale,
                                SQLPOINTER        pData,
                                SQLINTEGER        cbValueMax,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLCancel       (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLCloseCursor  (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLColAttributes (SQLHSTMT          hstmt,
                                 SQLSMALLINT       icol,
                                 SQLSMALLINT       fDescType,
                                 SQLCHAR          *rgbDesc,
                                 SQLINTEGER        cbDescMax,
                                 SQLINTEGER       *pcbDesc,
                                 SQLINTEGER       *pfDesc);

SQL_EXTERN SQLRETURN   SQLColumns    (SQLHSTMT       hstmt,
                               SQLCHAR        *szTableQualifier,
                               SQLSMALLINT    cbTableQualifier,
```

```
                                SQLCHAR        *szTableOwner,
                                SQLSMALLINT    cbTableOwner,
                                SQLCHAR        *szTableName,
                                SQLSMALLINT    cbTableName,
                                SQLCHAR        *szColumnName,
                                SQLSMALLINT    cbColumnName);

SQL_EXTERN SQLRETURN  SQLConnect    (SQLHDBC           hdbc,
                                SQLCHAR          *szDSN,
                                SQLSMALLINT       cbDSN,
                                SQLCHAR          *szUID,
                                SQLSMALLINT       cbUID,
                                SQLCHAR          *szAuthStr,
                                SQLSMALLINT       cbAuthStr);

SQL_EXTERN SQLRETURN  SQLCopyDesc   (SQLHDESC   sDesc,
                                SQLHDESC   tDesc);

SQL_EXTERN SQLRETURN  SQLDataSources (SQLHENV       henv,
                                   SQLSMALLINT   fDirection,
                                   SQLCHAR      *szDSN,
                                   SQLSMALLINT   cbDSNMax,
                                   SQLSMALLINT  *pcbDSN,
                                   SQLCHAR      *szDescription,
                                   SQLSMALLINT   cbDescriptionMax,
                                   SQLSMALLINT  *pcbDescription);

SQL_EXTERN SQLRETURN  SQLDescribeCol (SQLHSTMT          hstmt,
                                SQLSMALLINT       icol,
                                SQLCHAR          *szColName,
                                SQLSMALLINT       cbColNameMax,
                                SQLSMALLINT      *pcbColName,
                                SQLSMALLINT      *pfSqlType,
                                SQLINTEGER       *pcbColDef,
                                SQLSMALLINT      *pibScale,
                                SQLSMALLINT      *pfNullable);

SQL_EXTERN SQLRETURN  SQLDescribeParam (SQLHSTMT         hstmt,
                                SQLSMALLINT       ipar,
                                SQLSMALLINT      *pfSqlType,
                                SQLINTEGER       *pcbColDef,
                                SQLSMALLINT      *pibScale,
                                SQLSMALLINT      *pfNullable);

SQL_EXTERN SQLRETURN  SQLDisconnect  (SQLHDBC             hdbc);

SQL_EXTERN SQLRETURN  SQLDriverConnect (SQLHDBC             hdbc,
                                   SQLPOINTER          hwnd,
                                   SQLCHAR      *szConnStrIn,
                                   SQLSMALLINT   cbConnStrin,
                                   SQLCHAR      *szConnStrOut,
                                   SQLSMALLINT  cbConnStrOutMax,
                                   SQLSMALLINT  *pcbConnStrOut,
                                   SQLSMALLINT fDriverCompletion);

SQL_EXTERN SQLRETURN  SQLEndTran     (SQLSMALLINT         htype,
                                SQLHENV              henv,
                                SQLSMALLINT          ctype);

SQL_EXTERN SQLRETURN  SQLError       (SQLHENV             henv,
                                SQLHDBC              hdbc,
                                SQLHSTMT             hstmt,
                                SQLCHAR          *szSqlState,
                                SQLINTEGER       *pfNativeError,
                                SQLCHAR          *szErrorMsg,
                                SQLSMALLINT       cbErrorMsgMax,
                                SQLSMALLINT      *pcbErrorMsg);
```

```
SQL_EXTERN SQLRETURN  SQLExecDirect  (SQLHSTMT            hstmt,
                                  SQLCHAR          *szSqlStr,
                                  SQLINTEGER        cbSqlStr);

SQL_EXTERN SQLRETURN  SQLExecute     (SQLHSTMT            hstmt);

SQL_EXTERN SQLRETURN  SQLExtendedFetch (SQLHSTMT            hstmt,
                                  SQLSMALLINT          fOrient,
                                  SQLINTEGER           fOffset,
                                  SQLINTEGER          *pcrow,
                                  SQLSMALLINT         *rgfRowStatus);

SQL_EXTERN SQLRETURN  SQLFetch       (SQLHSTMT            hstmt);

SQL_EXTERN SQLRETURN  SQLFetchScroll (SQLHSTMT            hstmt,
                                   SQLSMALLINT          fOrient,
                                   SQLINTEGER           fOffset);

SQL_EXTERN SQLRETURN  SQLForeignKeys (SQLHSTMT          hstmt,
                                  SQLCHAR          *szPkTableQualifier,
                                  SQLSMALLINT       cbPkTableQualifier,
                                  SQLCHAR          *szPkTableOwner,
                                  SQLSMALLINT       cbPkTableOwner,
                                  SQLCHAR          *szPkTableName,
                                  SQLSMALLINT       cbPkTableName,
                                  SQLCHAR          *szFkTableQualifier,
                                  SQLSMALLINT       cbFkTableQualifier,
                                  SQLCHAR          *szFkTableOwner,
                                  SQLSMALLINT       cbFkTableOwner,
                                  SQLCHAR          *szFkTableName,
                                  SQLSMALLINT       cbFkTableName);

SQL_EXTERN SQLRETURN  SQLFreeConnect (SQLHDBC             hdbc);

SQL_EXTERN SQLRETURN  SQLFreeEnv     (SQLHENV             henv);

SQL_EXTERN SQLRETURN  SQLFreeStmt    (SQLHSTMT            hstmt,
                                  SQLSMALLINT          fOption);

SQL_EXTERN SQLRETURN  SQLFreeHandle  (SQLSMALLINT          htype,
                                  SQLINTEGER           hndl);

SQL_EXTERN SQLRETURN  SQLGetCol      (SQLHSTMT            hstmt,
                                  SQLSMALLINT          icol,
                                  SQLSMALLINT          itype,
                                  SQLPOINTER           tval,
                                  SQLINTEGER           blen,
                                  SQLINTEGER          *olen);

SQL_EXTERN SQLRETURN  SQLGetConnectAttr (SQLHDBC             hdbc,
                                   SQLINTEGER           attr,
                                   SQLPOINTER           oval,
                                   SQLINTEGER           ilen,
                                   SQLINTEGER          *olen);

SQL_EXTERN SQLRETURN  SQLGetConnectOption (SQLHDBC             hdbc,
                                    SQLSMALLINT          iopt,
                                    SQLPOINTER           oval);

SQL_EXTERN SQLRETURN  SQLGetCursorName (SQLHSTMT            hstmt,
                                  SQLCHAR          *szCursor,
                                  SQLSMALLINT       cbCursorMax,
                                  SQLSMALLINT      *pcbCursor);

SQL_EXTERN SQLRETURN  SQLGetData     (SQLHSTMT          hstmt,
                                  SQLSMALLINT       icol,
```

```
                                SQLSMALLINT       fCType,
                                SQLPOINTER        rgbValue,
                                SQLINTEGER        cbValueMax,
                                SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN SQLGetDescField (SQLHDESC            hdesc,
                                SQLSMALLINT          rcdNum,
                                SQLSMALLINT          fieldID,
                                SQLPOINTER           fValue,
                                SQLINTEGER           fLength,
                                SQLINTEGER          *stLength);

SQL_EXTERN SQLRETURN SQLGetDescRec   (SQLHDESC            hdesc,
                                SQLSMALLINT          rcdNum,
                                SQLCHAR             *fname,
                                SQLSMALLINT          bufLen,
                                SQLSMALLINT         *sLength,
                                SQLSMALLINT         *sType,
                                SQLSMALLINT         *sbType,
                                SQLINTEGER          *fLength,
                                SQLSMALLINT         *fprec,
                                SQLSMALLINT         *fscale,
                                SQLSMALLINT         *fnull);

SQL_EXTERN SQLRETURN SQLGetDiagField (SQLSMALLINT         hType,
                                SQLINTEGER          hndl,
                                SQLSMALLINT         rcdNum,
                                SQLSMALLINT         diagID,
                                SQLPOINTER          dValue,
                                SQLSMALLINT         bLength,
                                SQLSMALLINT        *sLength);

SQL_EXTERN SQLRETURN SQLGetDiagRec   (SQLSMALLINT         hType,
                                SQLINTEGER          hndl,
                                SQLSMALLINT         rcdNum,
                                SQLCHAR            *SQLstate,
                                SQLINTEGER         *SQLcode,
                                SQLCHAR            *msgText,
                                SQLSMALLINT         bLength,
                                SQLSMALLINT        *SLength);

SQL_EXTERN SQLRETURN SQLGetEnvAttr  (SQLHENV    hEnv,
                               SQLINTEGER fAttribute,
                               SQLPOINTER pParam,
                               SQLINTEGER cbParamMax,
                               SQLINTEGER * pcbParam);

SQL_EXTERN SQLRETURN  SQLGetFunctions  (SQLHDBC        hdbc,
                                   SQLSMALLINT    fFunction,
                                   SQLSMALLINT    *pfExists);

SQL_EXTERN SQLRETURN  SQLGetInfo       (SQLHDBC        hdbc,
                                   SQLSMALLINT    fInfoType,
                                   SQLPOINTER     rgbInfoValue,
                                   SQLSMALLINT    cbInfoValueMax,
                                   SQLSMALLINT    *pcbInfoValue);

SQL_EXTERN SQLRETURN  SQLGetLength (SQLHSTMT       hstmt,
                                   SQLSMALLINT    locType,
                                   SQLINTEGER     locator,
                                   SQLINTEGER    *sLength,
                                   SQLINTEGER    *ind);

SQL_EXTERN SQLRETURN  SQLGetPosition (SQLHSTMT       hstmt,
                                   SQLSMALLINT    locType,
                                   SQLINTEGER     srceLocator,
                                   SQLINTEGER     srchLocator,
```

```
                                  SQLCHAR       *srchLiteral,
                                  SQLINTEGER     srchLiteralLen,
                                  SQLINTEGER     fPosition,
                                  SQLINTEGER    *located,
                                  SQLINTEGER    *ind);

SQL_EXTERN SQLRETURN  SQLGetStmtAttr   (SQLHSTMT       hstmt,
                                  SQLINTEGER     fAttr,
                                  SQLPOINTER     pvParam,
                                  SQLINTEGER     bLength,
                                  SQLINTEGER    *SLength);

SQL_EXTERN SQLRETURN  SQLGetStmtOption (SQLHSTMT       hstmt,
                                  SQLSMALLINT    fOption,
                                  SQLPOINTER     pvParam);

SQL_EXTERN SQLRETURN  SQLGetSubString (SQLHSTMT       hstmt,
                                  SQLSMALLINT    locType,
                                  SQLINTEGER     srceLocator,
                                  SQLINTEGER     fPosition,
                                  SQLINTEGER     length,
                                  SQLSMALLINT    tType,
                                  SQLPOINTER     rgbValue,
                                  SQLINTEGER     cbValueMax,
                                  SQLINTEGER    *StringLength,
                                  SQLINTEGER    *ind);

SQL_EXTERN SQLRETURN  SQLGetTypeInfo   (SQLHSTMT       hstmt,
                                  SQLSMALLINT    fSqlType);

SQL_EXTERN SQLRETURN  SQLLanguages   (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLMoreResults (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN  SQLNativeSql   (SQLHDBC           hdbc,
                                   SQLCHAR          *szSqlStrIn,
                                   SQLINTEGER        cbSqlStrIn,
                                   SQLCHAR          *szSqlStr,
                                   SQLINTEGER        cbSqlStrMax,
                                   SQLINTEGER       *pcbSqlStr);

SQL_EXTERN SQLRETURN  SQLNumParams (SQLHSTMT            hstmt,
                                  SQLSMALLINT      *pcpar);

SQL_EXTERN SQLRETURN  SQLNumResultCols (SQLHSTMT        hstmt,
                                  SQLSMALLINT      *pccol);

SQL_EXTERN SQLRETURN  SQLParamData   (SQLHSTMT          hstmt,
                                SQLPOINTER          *Value);

SQL_EXTERN SQLRETURN  SQLParamOptions (SQLHSTMT         hstmt,
                                SQLINTEGER          crow,
                                SQLINTEGER          *pirow);

SQL_EXTERN SQLRETURN  SQLPrepare     (SQLHSTMT          hstmt,
                                SQLCHAR           *szSqlStr,
                                SQLSMALLINT       cbSqlStr);

SQL_EXTERN SQLRETURN  SQLPrimaryKeys (SQLHSTMT          hstmt,
                                SQLCHAR           *szTableQualifier,
                                SQLSMALLINT       cbTableQualifier,
                                SQLCHAR           *szTableOwner,
                                SQLSMALLINT       cbTableOwner,
                                SQLCHAR           *szTableName,
                                SQLSMALLINT       cbTableName);

SQL_EXTERN SQLRETURN  SQLProcedureColumns (SQLHSTMT          hstmt,
```

```
                                SQLCHAR          *szProcQualifier,
                                SQLSMALLINT       cbProcQualifier,
                                SQLCHAR          *szProcOwner,
                                SQLSMALLINT       cbProcOwner,
                                SQLCHAR          *szProcName,
                                SQLSMALLINT       cbProcName,
                                SQLCHAR          *szColumnName,
                                SQLSMALLINT       cbColumnName);

SQL_EXTERN SQLRETURN  SQLProcedures (SQLHSTMT          hstmt,
                                SQLCHAR          *szProcQualifier,
                                SQLSMALLINT       cbProcQualifier,
                                SQLCHAR          *szProcOwner,
                                SQLSMALLINT       cbProcOwner,
                                SQLCHAR          *szProcName,
                                SQLSMALLINT       cbProcName);

SQL_EXTERN SQLRETURN  SQLPutData     (SQLHSTMT            hstmt,
                                SQLPOINTER           Data,
                                SQLINTEGER           SLen);

SQL_EXTERN SQLRETURN  SQLReleaseEnv     (SQLHENV           henv);

SQL_EXTERN SQLRETURN  SQLRowCount    (SQLHSTMT           hstmt,
                                SQLINTEGER         *pcrow);

SQL_EXTERN SQLRETURN  SQLSetConnectAttr   (SQLHDBC        hdbc,
                                     SQLINTEGER     attrib,
                                     SQLPOINTER     vParam,
                                     SQLINTEGER     inlen);

SQL_EXTERN SQLRETURN  SQLSetConnectOption (SQLHDBC        hdbc,
                                     SQLSMALLINT    fOption,
                                     SQLPOINTER     vParam);

SQL_EXTERN SQLRETURN  SQLSetCursorName (SQLHSTMT            hstmt,
                                   SQLCHAR          *szCursor,
                                   SQLSMALLINT       cbCursor);

SQL_EXTERN SQLRETURN  SQLSetDescField  (SQLHDESC          hdesc,
                                   SQLSMALLINT       rcdNum,
                                   SQLSMALLINT       fID,
                                   SQLPOINTER        Value,
                                   SQLINTEGER        buffLen);

SQL_EXTERN SQLRETURN  SQLSetDescRec    (SQLHDESC          hdesc,
                                   SQLSMALLINT       rcdNum,
                                   SQLSMALLINT       Type,
                                   SQLSMALLINT       subType,
                                   SQLINTEGER        fLength,
                                   SQLSMALLINT       fPrec,
                                   SQLSMALLINT       fScale,
                                   SQLPOINTER        Value,
                                   SQLINTEGER       *sLength,
                                   SQLSMALLINT      *indic);

SQL_EXTERN SQLRETURN  SQLSetEnvAttr( SQLHENV hEnv,
                                SQLINTEGER fAttribute,
                                SQLPOINTER pParam,
                                SQLINTEGER cbParam);

SQL_EXTERN SQLRETURN  SQLSetParam      (SQLHSTMT            hstmt,
                                   SQLSMALLINT       ipar,
                                   SQLSMALLINT       fCType,
                                   SQLSMALLINT       fSqlType,
                                   SQLINTEGER        cbColDef,
                                   SQLSMALLINT       ibScale,
```

```
                                    SQLPOINTER        rgbValue,
                                    SQLINTEGER       *pcbValue);

SQL_EXTERN SQLRETURN  SQLSetStmtAttr (SQLHSTMT       hstmt,
                                    SQLINTEGER     fAttr,
                                    SQLPOINTER     pParam,
                                    SQLINTEGER     vParam);

SQL_EXTERN SQLRETURN  SQLSetStmtOption (SQLHSTMT          hstmt,
                                    SQLSMALLINT    fOption,
                                    SQLPOINTER     vParam);

SQL_EXTERN SQLRETURN SQLSpecialColumns (SQLHSTMT       hstmt,
                                    SQLSMALLINT    fColType,
                                    SQLCHAR        *szTableQual,
                                    SQLSMALLINT    cbTableQual,
                                    SQLCHAR        *szTableOwner,
                                    SQLSMALLINT    cbTableOwner,
                                    SQLCHAR        *szTableName,
                                    SQLSMALLINT    cbTableName,
                                    SQLSMALLINT    fScope,
                                    SQLSMALLINT    fNullable);

SQL_EXTERN SQLRETURN  SQLStatistics    (SQLHSTMT       hstmt,
                                    SQLCHAR        *szTableQualifier,
                                    SQLSMALLINT    cbTableQualifier,
                                    SQLCHAR        *szTableOwner,
                                    SQLSMALLINT    cbTableOwner,
                                    SQLCHAR        *szTableName,
                                    SQLSMALLINT    cbTableName,
                                    SQLSMALLINT    fUnique,
                                    SQLSMALLINT    fres);

SQL_EXTERN SQLRETURN SQLTables        (SQLHSTMT       hstmt,
                                   SQLCHAR        *szTableQualifier,
                                   SQLSMALLINT    cbTableQualifier,
                                   SQLCHAR        *szTableOwner,
                                   SQLSMALLINT    cbTableOwner,
                                   SQLCHAR        *szTableName,
                                   SQLSMALLINT    cbTableName,
                                   SQLCHAR        *szTableType,
                                   SQLSMALLINT    cbTableType);

SQL_EXTERN SQLRETURN  SQLTransact      (SQLHENV             henv,
                                    SQLHDBC             hdbc,
                                    SQLSMALLINT       fType);


#define FAR
#define  SQL_SQLSTATE_SIZE         5   /* size of SQLSTATE, not including
                                          null terminating byte              */
#define  SQL_MAX_DSN_LENGTH       18   /* maximum data source name size      */
#define  SQL_MAX_ID_LENGTH        18   /* maximum identifier name size,
                                          e.g. cursor names                  */

#define SQL_MAX_STMT_SIZ       32767   /* Maximum statement size             */

#define SQL_MAXRECL            32766   /* Maximum record length              */

#define SQL_SMALL_LENGTH       2       /* Size of a SMALLINT                 */
#define SQL_MAXSMALLVAL        32767   /* Maximum value of a SMALLINT        */
#define SQL_MINSMALLVAL (-(SQL_MAXSMALLVAL)-1) /* Minimum value of a SMALLINT */
#define SQL_INT_LENGTH         4       /* Size of an INTEGER                 */
#define SQL_MAXINTVAL          2147483647 /* Maximum value of an INTEGER     */
#define SQL_MININTVAL (-(SQL_MAXINTVAL)-1)/* Minimum value of an INTEGER     */
#define SQL_FLOAT_LENGTH       8       /* Size of a FLOAT                    */
#define SQL_DEFDEC_PRECISION   5       /* Default precision for DECIMAL      */
```

```
#define SQL_DEFDEC_SCALE       0       /* Default scale for DECIMAL          */
#define SQL_MAXDECIMAL         31      /* Maximum scale/prec. for DECIMAL    */
#define SQL_DEFCHAR            1       /* Default length for a CHAR          */
#define SQL_DEFWCHAR           1       /* Default length for a wchar_t       */
#define SQL_MAXCHAR            32766   /* Maximum length of a CHAR           */
#define SQL_MAXLSTR            255     /* Maximum length of an LSTRING       */
#define SQL_MAXVCHAR           32740   /* Maximum length of a         */
                                       /* VARCHAR                            */
#define SQL_MAXVGRAPH          16370   /* Maximum length of a VARGRAPHIC     */
#define SQL_MAXBLOB            15728640 /* Max. length of a BLOB host var   */
#define SQL_MAXCLOB            15728640 /* Max. length of a CLOB host var   */
#define SQL_MAXDBCLOB          7864320 /* Max. length of an DBCLOB host    */
                                       /* var                                */
#define SQL_LONGMAX            32740   /* Maximum length of a LONG VARCHAR   */
#define SQL_LONGGRMAX          16370   /* Max. length of a LONG VARGRAPHIC   */
#define SQL_LVCHAROH           26      /* Overhead for LONG VARCHAR in       */
                                       /* record                             */
#define SQL_LOBCHAROH          312     /* Overhead for LOB in record         */
#define SQL_BLOB_MAXLEN        15728640 /* BLOB maximum length, in bytes    */
#define SQL_CLOB_MAXLEN        15728640 /* CLOB maximum length, in chars    */
#define SQL_DBCLOB_MAXLEN      7864320 /* maxlen for dbcs lobs             */
#define SQL_TIME_LENGTH        3       /* Size of a TIME field               */
#define SQL_TIME_STRLEN        8       /* Size of a TIME field output        */
#define SQL_TIME_MINSTRLEN     5       /* Size of a non-USA TIME field       */
                                       /* output without seconds             */
#define SQL_DATE_LENGTH        4       /* Size of a DATE field               */
#define SQL_DATE_STRLEN        10      /* Size of a DATE field output        */
#define SQL_STAMP_LENGTH       10      /* Size of a TIMESTAMP field          */
#define SQL_STAMP_STRLEN       26      /* Size of a TIMESTAMP field output   */
#define SQL_STAMP_MINSTRLEN    19      /* Size of a TIMESTAMP field output   */
                                       /* without microseconds               */
#define SQL_BOOLEAN_LENGTH     1       /* Size of a BOOLEAN field            */
#define SQL_IND_LENGTH         2       /* Size of an indicator value         */

#define SQL_MAX_PNAME_LENGTH   254     /* Max size of Stored Proc Name       */
#define SQL_LG_IDENT           18      /* Maximum length of Long Identifer   */
#define SQL_SH_IDENT           8       /* Maximum length of Short Identifer  */
#define SQL_MN_IDENT           1       /* Minimum length of Identifiers      */
#define SQL_MAX_VAR_NAME       30      /* Max size of Host Variable Name     */
#define SQL_KILO_VALUE         1024    /* # of bytes in a kilobyte           */
#define SQL_MEGA_VALUE         1048576 /* # of bytes in a megabyte           */
#define SQL_GIGA_VALUE         1073741824 /* # of bytes in a gigabyte        */

/* SQL extended data types (negative means unsupported) */
#define  SQL_TINYINT            -6
#define  SQL_BIT                -7

/* C data type to SQL data type mapping */
#define  SQL_C_CHAR       SQL_CHAR      /* CHAR, VARCHAR, DECIMAL, NUMERIC */
#define  SQL_C_LONG       SQL_INTEGER   /* INTEGER                         */
#define  SQL_C_SHORT      SQL_SMALLINT  /* SMALLINT                        */
#define  SQL_C_FLOAT      SQL_REAL      /* REAL                            */
#define  SQL_C_DOUBLE     SQL_DOUBLE    /* FLOAT, DOUBLE                   */
#define  SQL_C_DATE       SQL_DATE      /* DATE                            */
#define  SQL_C_TIME       SQL_TIME      /* TIME                            */
#define  SQL_C_TIMESTAMP  SQL_TIMESTAMP /* TIMESTAMP                       */
#define  SQL_C_BINARY     SQL_BINARY    /* BINARY, VARBINARY               */
#define  SQL_C_BIT        SQL_BIT
#define  SQL_C_TINYINT    SQL_TINYINT
#define  SQL_C_BIGINT     SQL_BIGINT
#define  SQL_C_DBCHAR     SQL_DBCLOB
#define  SQL_C_WCHAR      SQL_WCHAR     /* UNICODE                         */
#define  SQL_C_DATETIME   SQL_DATETIME  /* DATETIME                        */
#define  SQL_C_BLOB           SQL_BLOB
#define  SQL_C_CLOB           SQL_CLOB
#define  SQL_C_DBCLOB         SQL_DBCLOB
#define  SQL_C_BLOB_LOCATOR   SQL_BLOB_LOCATOR
```

```
#define  SQL_C_CLOB_LOCATOR   SQL_CLOB_LOCATOR
#define  SQL_C_DBCLOB_LOCATOR SQL_DBCLOB_LOCATOR



#define SQL_WARN_VAL_TRUNC                  "01004"


#endif /* SQL_H_SQLCLI */
```

附录 C. 示例 DB2 UDB CLI 应用程序代码列表

本附录部分给出整本书中使用的示例的完整代码列表。

尚未在示例中实现详细的错误检查。

参见:

- 『示例: 嵌入式 SQL 和等价的 DB2 UDB CLI 函数调用』
- 第 264 页的『示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用』

示例: 嵌入式 SQL 和等价的 DB2 UDB CLI 函数调用

此示例在注释中显示嵌入式语句以及等价的 DB2 UDB CLI 函数调用。

有关代码示例的信息, 参见第 viii 页的『代码不保证声明信息』。

```
/************************************************************************
** file = embedded.c
**
** Example of executing an SQL statement using CLI.
** The equivalent embedded SQL statements are shown in comments.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLSetParam          SQLTransact
**        SQLError             SQLExecDirect
**
************************************************************************/
#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#ifndef NULL
#define NULL   0
#endif

int print_err (SQLHDBC    hdbc,
               SQLHSTMT   hstmt);

int main ()
{
    SQLHENV        henv;
    SQLHDBC        hdbc;
    SQLHSTMT       hstmt;

    SQLCHAR     server[] = "sample";
    SQLCHAR     uid[30];
    SQLCHAR     pwd[30];

    SQLINTEGER     id;
    SQLCHAR        name[51];
    SQLINTEGER     namelen, intlen;
    SQLSMALLINT    scale;

    scale = 0;
```

```
/* EXEC SQL CONNECT TO :server USER :uid USING :authentication_string; */
SQLAllocEnv (&henv);                /* allocate an environment handle */

SQLAllocConnect (henv, &hdbc);      /* allocate a connection handle   */


/* Connect to database indicated by "server" variable with             */
/*    authorization-name given in "uid", authentication-string given   */
/*    in "pwd". Note server, uid, and pwd contain null-terminated      */
/*    strings, as indicated by the 3 input lengths set to SQL_NTS      */
if (SQLConnect (hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS)
              != SQL_SUCCESS)
    return (print_err (hdbc, SQL_NULL_HSTMT));

SQLAllocStmt (hdbc, &hstmt);        /* allocate a statement handle    */


/* EXEC SQL CREATE TABLE NAMEID (ID integer, NAME varchar(50));        */
{
   SQLCHAR create[] = "CREATE TABLE NAMEID (ID integer, NAME varchar(50))";

/* execute the sql statement                                           */
   if (SQLExecDirect (hstmt, create, SQL_NTS) != SQL_SUCCESS)
       return (print_err (hdbc, hstmt));
}


/* EXEC SQL COMMIT WORK;                                               */
SQLTransact (henv, hdbc, SQL_COMMIT);          /* commit create table */


/* EXEC SQL INSERT INTO NAMEID VALUES ( :id, :name         */
{
   SQLCHAR insert[] = "INSERT INTO NAMEID VALUES (?, ?)";


/* show the use of SQLPrepare/SQLExecute method                        */
/* prepare the insert                                                  */

   if (SQLPrepare (hstmt, insert, SQL_NTS) != SQL_SUCCESS)
      return (print_err (hdbc, hstmt));

/* Set up the first input parameter "id"                               */
   intlen = sizeof (SQLINTEGER);
   SQLSetParam (hstmt, 1,
                SQL_C_LONG, SQL_INTEGER,
               (SQLINTEGER) sizeof (SQLINTEGER),
                scale, (SQLPOINTER) &id,
               (SQLINTEGER *) &intlen);

   namelen = SQL_NTS;
/* Set up the second input parameter "name"                            */
   SQLSetParam (hstmt, 2,
                SQL_C_CHAR, SQL_VARCHAR,
                50,
                scale, (SQLPOINTER) name,
                (SQLINTEGER *) &namelen);

/* now assign parameter values and execute the insert                 */
   id=500;
   strcpy (name, "Babbage");

   if (SQLExecute (hstmt) != SQL_SUCCESS)
      return (print_err (hdbc, hstmt));
}
```

```
    /* EXEC SQL COMMIT WORK;                                              */
    SQLTransact (henv, hdbc, SQL_COMMIT);        /* commit inserts        */


    /* EXEC SQL DECLARE c1 CURSOR FOR SELECT ID, NAME FROM NAMEID;        */
    /* EXEC SQL OPEN c1;                                                  */
    /* The application doesn't specify "declare c1 cursor for"            */
    {
       SQLCHAR 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;                    */
    /* Binding first column to output variable "id"                       */
    SQLBindCol (hstmt, 1,
                SQL_C_LONG, (SQLPOINTER) &id,
               (SQLINTEGER) sizeof (SQLINTEGER),
               (SQLINTEGER *) &intlen);

    /* Binding second column to output variable "name"                    */
    SQLBindCol (hstmt, 2,
                SQL_C_CHAR, (SQLPOINTER) name,
               (SQLINTEGER) sizeof (name),
                &namelen);

    SQLFetch (hstmt);                     /* now execute the fetch        */
    printf("Result of Select: id = %ld name = %s\n", id, name);

    /* finally, we should commit, discard hstmt, disconnect               */
    /* EXEC SQL COMMIT WORK;                                              */
    SQLTransact (henv, hdbc, SQL_COMMIT);  /* commit the transaction      */

    /* EXEC SQL CLOSE c1;                                                 */
    SQLFreeStmt (hstmt, SQL_DROP);         /* free the statement handle   */

    /* EXEC SQL DISCONNECT;                                               */
    SQLDisconnect (hdbc);                  /* disconnect from the database */

    SQLFreeConnect (hdbc);                 /* free the connection handle  */
    SQLFreeEnv (henv);                     /* free the environment handle */

    return (0);
}

int print_err (SQLHDBC    hdbc,
               SQLHSTMT   hstmt)
{
SQLCHAR        buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR        sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER     sqlcode;
SQLSMALLINT    length;


        while ( SQLError(SQL_NULL_HENV, hdbc, hstmt,
                  sqlstate,
                  &sqlcode,
                  buffer,
                  SQL_MAX_MESSAGE_LENGTH + 1,
                  &length) == SQL_SUCCESS )
        {
                 printf("SQLSTATE: %s Native Error Code: %ld\n",
                        sqlstate, sqlcode);
                 printf("%s \n", buffer);
```

```
                printf("---------------------------- \n");
         };

        return(SQL_ERROR);

}
```

示例: 交互式 SQL 和等价的 DB2 UDB CLI 函数调用

此示例显示交互式 SQL 语句的执行，并遵循第 7 页的第 2 章，『编写 DB2 UDB CLI 应用程序』中描述的流程。

有关代码示例的信息，参见第 viii 页的『代码不保证声明信息』。

```
/************************************************************************
** file = typical.c
**
** Example of executing interactive SQL statements, displaying result sets
** and simple transaction management.
**
** Functions used:
**
**        SQLAllocConnect      SQLFreeConnect
**        SQLAllocEnv          SQLFreeEnv
**        SQLAllocStmt         SQLFreeStmt
**        SQLConnect           SQLDisconnect
**
**        SQLBindCol           SQLFetch
**        SQLDescribeCol       SQLNumResultCols
**        SQLError             SQLRowCount
**        SQLExecDirect        SQLTransact
**
************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define  MAX_STMT_LEN 255
#define  MAXCOLS   100

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

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc);

int process_stmt(SQLHENV    henv,
                 SQLHDBC        hdbc,
                 SQLCHAR        *sqlstr);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error(SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt);

int check_error(SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt,
                SQLRETURN  frc);

void display_results(SQLHSTMT hstmt,
                     SQLSMALLINT nresultcols);
```

```
/*****************************************************************
** main
** - initialize
** - start a transaction
**   - get statement
**   - another statement?
** - COMMIT or ROLLBACK
** - another transaction?
** - terminate
*****************************************************************/
int main()
{
    SQLHENV     henv;
    SQLHDBC     hdbc;
    SQLCHAR     sqlstmt[MAX_STMT_LEN + 1]="";
    SQLCHAR     sqltrans[sizeof("ROLLBACK")];
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));

    printf("Enter an SQL statement to start a transaction(or 'q' to Quit):\n");
    gets(sqlstmt);

    while (sqlstmt[0] !='q')
    {
        while (sqlstmt[0] != 'q')
        {   rc = process_stmt(henv, hdbc, sqlstmt);
            if (rc == SQL_ERROR) return(SQL_ERROR);
            printf("Enter an SQL statement(or 'q' to Quit):\n");
            gets(sqlstmt);
        }

        printf("Enter 'c' to COMMIT or 'r' to ROLLBACK the transaction\n");
        fgets(sqltrans, sizeof("ROLLBACK"), stdin);

        if (sqltrans[0] == 'c')
        {
          rc = SQLTransact (henv, hdbc, SQL_COMMIT);
          if (rc == SQL_SUCCESS)
            printf ("Transaction commit was successful\n");
          else
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
        }

        if (sqltrans[0] == 'r')
        {
          rc = SQLTransact (henv, hdbc, SQL_ROLLBACK);
          if (rc == SQL_SUCCESS)
            printf ("Transaction roll back was successful\n");
          else
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
        }

        printf("Enter an SQL statement to start a transaction or 'q' to quit\n");
        gets(sqlstmt);
    }

    terminate(henv, hdbc);

    return (SQL_SUCCESS);
}/* end main */

/*****************************************************************
** process_stmt
** - allocates a statement handle
** - executes the statement
```

```
** - determines the type of statement
**   - if there are no result columns, therefore non-select statement
**      - if rowcount > 0, assume statement was UPDATE, INSERT, DELETE
**     else
**      - assume a DDL, or Grant/Revoke statement
**   else
**      - must be a select statement.
**      - display results
** - frees the statement handle
*****************************************************************/

int process_stmt (SQLHENV    henv,
                  SQLHDBC    hdbc,
                  SQLCHAR    *sqlstr)
{
SQLHSTMT        hstmt;
SQLSMALLINT     nresultcols;
SQLINTEGER      rowcount;
SQLRETURN       rc;


    SQLAllocStmt (hdbc, &hstmt);       /* allocate a statement handle */

    /* execute the SQL statement in "sqlstr"    */

    rc = SQLExecDirect (hstmt, sqlstr, SQL_NTS);
    if (rc != SQL_SUCCESS)
        if (rc == SQL_NO_DATA_FOUND) {
            printf("\nStatement executed without error, however,\n");
            printf("no data was found or modified\n");
            return (SQL_SUCCESS);
        }
        else
            check_error (henv, hdbc, hstmt, rc);

    SQLRowCount (hstmt, &rowcount);
    rc = SQLNumResultCols (hstmt, &nresultcols);
    if (rc != SQL_SUCCESS)
      check_error (henv, hdbc, hstmt, rc);

    /* determine statement type */
    if (nresultcols == 0) /* statement is not a select statement */
    {
        if (rowcount > 0 ) /* assume statement is UPDATE, INSERT, DELETE */
        {
            printf ("Statement executed, %ld rows affected\n", rowcount);
        }
      else  /* assume statement is GRANT, REVOKE or a DLL statement */
        {
              printf ("Statement completed successful\n");
        }
    }
    else /* display the result set */
    {
        display_results(hstmt, nresultcols);
    } /* end determine statement type */

    SQLFreeStmt (hstmt, SQL_DROP );       /* free statement handle */

    return (0);
}/* end process_stmt */

/*****************************************************************
** initialize
**  - allocate environment handle
**  - allocate connection handle
**  - prompt for server, user id, & password
```

```
**  - connect to server
*********************************************************************/

int initialize(SQLHENV *henv,
               SQLHDBC *hdbc)
{
SQLCHAR     server[18],
            uid[10],
            pwd[10];
SQLRETURN   rc;

    rc = SQLAllocEnv (henv);         /* allocate an environment handle   */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    rc = SQLAllocConnect (*henv, hdbc);  /* allocate a connection handle     */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
    else
    {   rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
}/* end initialize */

/*********************************************************************
** terminate
**  - disconnect
**  - free connection handle
**  - free environment handle
*********************************************************************/
int terminate(SQLHENV henv,
              SQLHDBC hdbc)
{
SQLRETURN   rc;

    rc = SQLDisconnect (hdbc);          /* disconnect from database      */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeConnect (hdbc);         /* free connection handle        */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeEnv (henv);             /* free environment handle       */
    if (rc != SQL_SUCCESS )
        print_error (henv, SQL_NULL_HDBC, SQL_NULL_HSTMT);

}/* end terminate */

/*********************************************************************
** display_results - displays the selected character fields
**
**  - for each column
**      - get column name
**      - bind column
```

```
**  - display column headings
**  - fetch each row
**      - if value truncated, build error message
**      - if column null, set value to "NULL"
**      - display row
**      - print truncation message
**  - free local storage
**
****************************************************************/
void display_results(SQLHSTMT hstmt,
                SQLSMALLINT nresultcols)
{
SQLCHAR         colname[32];
SQLSMALLINT     coltype[MAXCOLS];
SQLSMALLINT     colnamelen;
SQLSMALLINT     nullable;
SQLINTEGER      collen[MAXCOLS];
SQLSMALLINT     scale;
SQLINTEGER      outlen[MAXCOLS];
SQLCHAR *       data[MAXCOLS];
SQLCHAR         errmsg[256];
SQLRETURN       rc;
SQLINTEGER      i;
SQLINTEGER      displaysize;

    for (i = 0; i < nresultcols; i++)
    {
        SQLDescribeCol (hstmt, i+1, colname, sizeof (colname),
        &colnamelen, &coltype[i], &collen[i], &scale, &nullable);

        /* get display length for column */
        SQLColAttributes (hstmt, i+1, SQL_DESC_PRECISION, NULL, 0     ,
            NULL, &displaysize);

        /* set column length to max of display length, and column name
           length.  Plus one byte for null terminator       */
        collen[i] = max(displaysize, collen[i]);
        collen[i] = max(collen[i], strlen((char *) colname) ) + 1;

        printf ("%-*.*s", collen[i], collen[i], colname);

        /* allocate memory to bind column                              */
        data[i] = (SQLCHAR *) malloc (collen[i]);

        /* bind columns to program vars, converting all types to CHAR */
        SQLBindCol (hstmt, i+1, SQL_C_CHAR, data[i], collen[i], &outlen[i]);
    }
    printf("\n");

    /* display result rows                                            */
    while ((rc = SQLFetch (hstmt)) != SQL_NO_DATA_FOUND)
    {
        errmsg[0] = '\0';
        for (i = 0; i < nresultcols; i++)
        {
            /* Build a truncation message for any columns truncated */
            if (outlen[i] >= collen[i])
            {    sprintf ((char *) errmsg + strlen ((char *) errmsg),
                        "%d chars truncated, col %d\n",
                         outlen[i]-collen[i]+1, i+1);
            }
            if (outlen[i] == SQL_NULL_DATA)
                printf ("%-*.*s", collen[i], collen[i], "NULL");
            else
                printf ("%-*.*s", collen[i], collen[i], data[i]);
        } /* for all columns in this row  */
```

```
        printf ("\n%s", errmsg);  /* print any truncation messages    */
    } /* while rows to fetch */

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

}/* end display_results

/************************************************************************
** SUPPORT FUNCTIONS
**  - print_error     - call SQLError(), display SQLSTATE and message
**  - check_error     - call print_error
**                    - check severity of Return Code
**                    - rollback & exit if error, continue if warning
************************************************************************/

/************************************************************************/
int print_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt)
{
SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER  sqlcode;
SQLSMALLINT length;

    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                     SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR *****\n");
        printf("         SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };
    return;
}

/************************************************************************/
int check_error (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLHSTMT   hstmt,
                 SQLRETURN  frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
        else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
        exit(frc);
        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
```

```
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);

}
```

附录 D. 以服务器方式运行 DB2 UDB CLI

参见『为何要以 SQL 服务器方式运行 DB2 UDB CLI』。

为何要以 SQL 服务器方式运行 DB2 UDB CLI

要以 SQL 服务器方式运行的原因是许多应用程序需要充当数据库服务器。这表示单个作业将代替多个用户来执行 SQL 请求。如果不使用 SQL 服务器方式，则应用程序可能会遇到下列三项限制中的其中一项或多项:

1. 对于单个作业，每个激活组只能有一个落实事务。
2. 单个作业只能连接一次 RDB。
3. 无论在连接上传入了什么用户标识，所有 SQL 语句都在作业的用户概要文件下运行。

通过将所有 SQL 语句路由至不同的作业，SQL 服务器方式避开了这些限制。每个连接都在它自己的作业中运行。系统使用 QSYSWRK 子系统中的预启动作业来最大程度地缩短每个连接的启动时间。由于对 SQLConnect 的每个调用都可以接受不同的用户概要文件，所以每个作业还有它自己的落实事务。在执行 SQLDisconnect 之后，将重设作业，并将其放回到可用作业的池中。

有关以 SQL 服务器方式运行 DB2 UDB CLI 的更多信息，参见:

- 『以 SQL 服务器方式启动 DB2 UDB CLI』
- 第 272 页的『关于以服务器方式运行 DB2 UDB CLI 的限制』

以 SQL 服务器方式启动 DB2 UDB CLI

可以通过两种方法来使作业处于 SQL 服务器方式:

1. 最有可能的情况是使用 CLI 函数 SQLSetEnvAttr。由于 CLI 应用程序已使用多个连接句柄这一概念，所以 SQL 服务器方式最适合于 CLI 应用程序。请在分配 CLI 环境之后立即设置此方式。并且，在设置此方式之前，作业一定不能运行任何 SQL 或启动落实控制。如果存在那些情况的任何之一，则方式就不会更改为服务器方式，而 SQL 将继续以“直接插入”方式运行。

   示例

   ```
   .
   .
   SQLAllocEnv(&henv);
   long  attr;
   attr  =  SQL_TRUE
   SQLSetEnvAttr(henv,SQL_ATTR_SERVER_MODE,&attr,0);
   SQLAllocConnect(henv,&hdbc);
   .
   .
   ```

2. 第二种设置服务器方式的方法是使用“更改作业”（QWTCHGJB）API。有关 QWTCHGJB API 的完整描述，请参考“iSeries 信息中心”中的 API 主题。

在设置 SQL 服务器方式之后，所有 SQL 连接和 SQL 语句都将以服务器方式运行。不能来回切换。作业一旦进入服务器方式就不能启动落实控制，并且不能使用“交互式 SQL”。

关于以服务器方式运行 DB2 UDB CLI 的限制

- 作业必须在刚刚开始进行处理时（在执行任何其它操作之前）设置服务器方式。对于仅供 CLI 用户使用的作业，它们必须使用 SQLSetEnvAttr 调用来打开服务器方式。记住，在 SQLAllocEnv 之后但在任何其它调用之前执行此操作。在打开服务器方式之后，就不能将其关闭。
- 所有 SQL 函数都在预启动作业和落实控制中运行。无论是在进入服务器方式之前还是之后，不要在起始作业中启动落实控制。
- 由于在预启动作业中处理 SQL，所以对起始作业中的特定更改不敏感。这包括对库列表、作业优先级以及消息记录等的更改。预启动对起始作业中的 CCSID 值的更改敏感，其原因在于这可能会影响将数据映射回用户的程序的方式。
- 当以服务器方式运行时，应用程序必须使用 SQL 落实和回滚，或者以嵌入方式使用，或者由 SQL CLI 使用。由于没有在起始作业中运行落实控制，所以应用程序不能使用 CL 命令。作业在断开连接之前必须发出 COMMIT；否则将发生隐式的 ROLLBACK。
- 不可能在处于服务器方式的作业中使用交互式 SQL。当处于服务器方式时，使用 STRSQL 将导致 SQL6141 消息。
- 当处于服务器方式时，也不可能执行 SQL 编译。在运行已编译的 SQL 程序时可使用服务器方式，但一定不能为编译打开此方式。如果作业处于服务器方式，则编译将失败。
- SQLDataSources 在它不需要连接句柄就能够运行这一方面具有独特性。当处于服务器方式时，程序必须已完成与本地数据库的连接（在使用 SQLDataSources 之前）。由于使用 DataSources 来查找连接的 RDB 的名称，所以 IBM 支持在 SQLConnect 上对 RDB 名传送空指针。这将获取本地连接。这样就有可能编写出事先不了解系统名的类属程序。
- 当通过 CLI 执行落实和回滚时，对 SQLEndTran 和 SQLTransact 的调用必须包含连接句柄。当不是以服务器方式运行时，可以省略连接句柄来落实任何内容。然而，在服务器方式下，由于每个连接（或线程）都有自己的事务作用域限定，并不支持这样做。
- 当以 SQL 服务器方式运行时，不建议在线程之间共享连接句柄。这是因为一个线程可能会覆盖另一个线程仍要处理的返回数据或错误信息。

索引

中国印刷