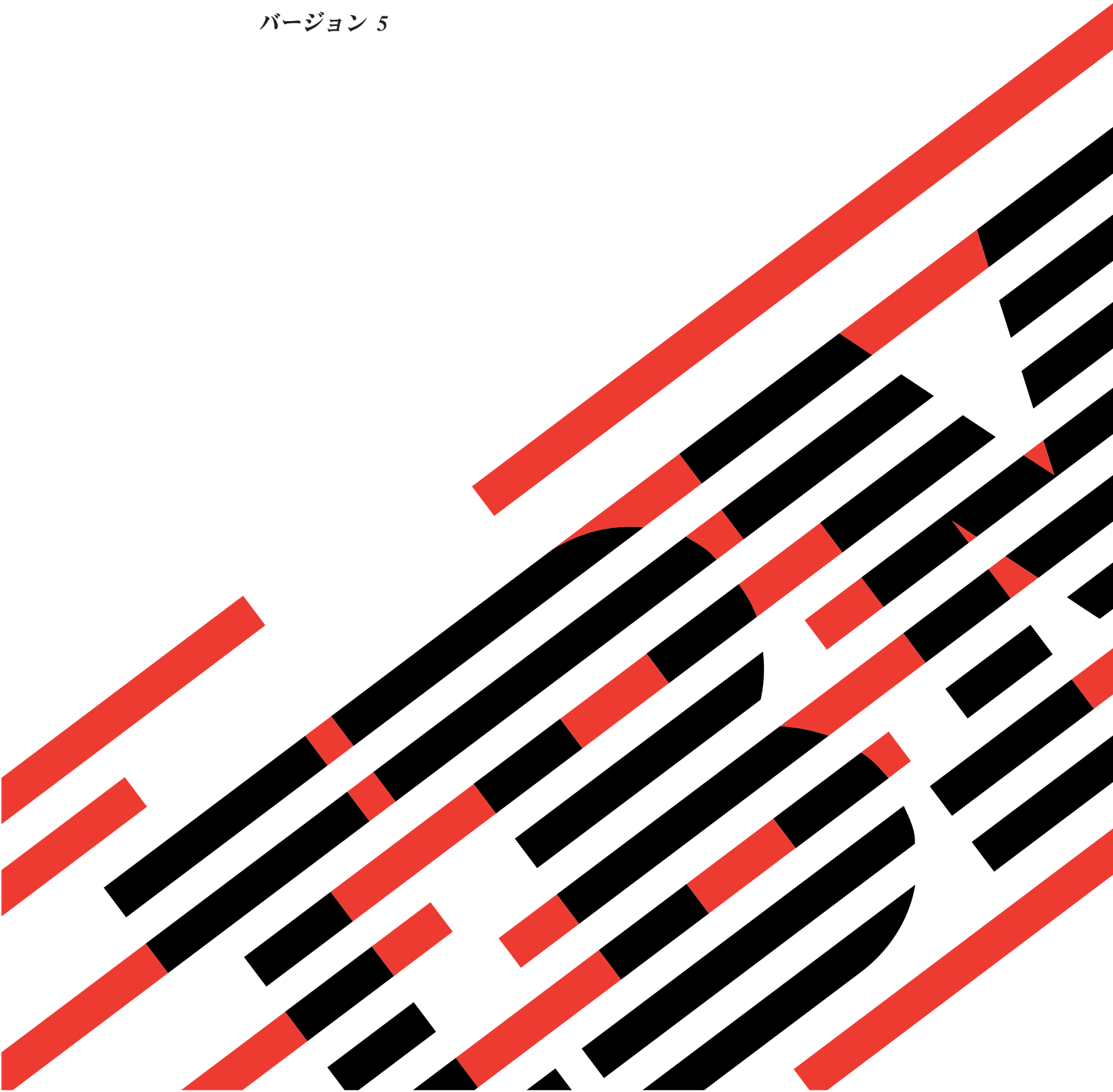
IBM

eserver

iSeries

DB2 Universal Database for iSeries
SQL 呼び出しレベル・インターフェース (ODBC)

バージョン 5



IBM

@server

iSeries

DB2 Universal Database for iSeries
SQL 呼び出しレベル・インターフェース (ODBC)

バージョン 5

目次

DB2 Universal Database for iSeries SQL 呼び出しレベル・インターフェース (ODBC) について vii
DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) の対象読者. vii
DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) V5R2 の新機能 vii
コードの特記事項情報 viii

第 1 章 CLI の紹介 1
DB2 UDB CLI の背景情報 1
呼び出しレベル・インターフェース 1
DB2 UDB CLI と組み込み SQL との相違. 4
組み込み SQL の代わりに DB2 UDB CLI を使用する利点 6
DB2 UDB CLI、動的 SQL、および静的 SQL のどれがよいか 6

第 2 章 DB2 UDB CLI アプリケーションの作成 7
DB2 UDB CLI アプリケーションでの初期設定と終了のタスク 8
例: DB2 UDB CLI アプリケーションでの初期設定と接続 9
DB2 UDB CLI アプリケーションでのトランザクション処理 10
DB2 UDB CLI アプリケーションでのステートメント・ハンドルの割り振り. 12
DB2 UDB CLI アプリケーションでの作成タスクと実行タスク. 12
DB2 UDB CLI アプリケーションでの処理結果 13
DB2 UDB CLI アプリケーションでのステートメント・ハンドルの解放 15
DB2 UDB CLI アプリケーションでのコミットまたはロールバック 15
DB2 UDB CLI アプリケーションでの診断 16
DB2 UDB CLI アプリケーションでの戻りコード. 16
DB2 UDB CLI SQLSTATE 17
DB2 UDB CLI の関数でのデータ・タイプとデータ変換 17
DB2 UDB CLI 関数でのその他の C データ・タイプ 18
DB2 UDB CLI 関数でのデータ変換. 19
DB2 UDB CLI 関数でのストリング引き数の処理. 20
DB2 UDB CLI 関数でのストリング引き数の長さ. 20
DB2 UDB CLI 関数でのストリングの切り捨て 21
DB2 UDB CLI 関数でのストリングの解釈 21

第 3 章 DB2 UDB CLI の関数 23
SQLAllocConnect - 接続の割り振り 27
SQLAllocEnv - 環境ハンドルの割り振り 30
SQLAllocHandle - ハンドルの割り振り. 33
SQLAllocStmt - ステートメント・ハンドルの割り振り 35
SQLBindCol - アプリケーション・プログラム変数に対する列のバインド 37
SQLBindFileToCol - LOB 列に対する LOB ファイル参照のバインド 42
SQLBindFileToParam - LOB パラメーターに対する LOB ファイル参照のバインド 45
SQLBindParam - パラメーター・マーカーに対するバッファーのバインド. 48
SQLBindParameter - バッファーに対するパラメーター・マーカーのバインド. 53
SQLCancel - ステートメントの取り消し 61
SQLCloseCursor - カーソル・ステートメントのクローズ 62
SQLColAttributes - 列属性 63
| SQLColumnPrivileges - 表の列に関連した特権の入手. 68
SQLColumns - 表の列情報の入手. 71
SQLConnect - データ・ソースへの接続 75
SQLCopyDesc - 記述ステートメントのコピー 78

SQLDataSources - データ・ソース・リストの入手 79
SQLDescribeCol - 列属性の記述 83
SQLDescribeParam - パラメーター・マーカーの記述を戻す 87
SQLDisconnect - データ・ソースからの切断. 90
SQLDriverConnect - (拡張) データ・ソースへの接続. 92
SQLEndTran - トランザクションのコミットまたはロールバック 96
SQLError - エラー情報の検索 98
SQLExecDirect - ステートメントの直接実行 101
SQLExecute - ステートメントの実行 103
SQLExtendedFetch - 行配列の取り出し 105
SQLFetch - 次のデータ行 108
SQLFetchScroll - スクロール可能カーソルからの取り出し 114
SQLForeignKeys - 外部キー列リストの入手 116
SQLFreeConnect - 接続ハンドルの解放 121
SQLFreeEnv - 環境ハンドルの解放. 123
SQLFreeHandle - ハンドルの解放 125
SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット). 127
SQLGetCol - 結果セットの行での 1 つの列の検索 130
SQLGetConnectAttr - 接続属性の値の取得 136
SQLGetConnectOption - 接続オプションの現行設定を戻す 138
SQLGetCursorName - カーソル名の取得 139
SQLGetData - 列のデータの取得 143
SQLGetDescField - 記述子フィールドの取得 144
SQLGetDescRec - 記述子レコードの取得 147
SQLGetDiagField - 診断情報 (拡張可能) を戻す 149
SQLGetDiagRec - 診断情報 (短縮型) を戻す 152
SQLGetEnvAttr - 環境属性の現行設定を戻す 155
SQLGetFunctions - 関数の取得 156
SQLGetInfo - 一般情報の取得. 159
SQLGetLength - ストリング値の長さの検索 174
SQLGetPosition - ストリングの開始位置を戻す 176
SQLGetStmtAttr - ステートメント属性の値の取得 179
SQLGetStmtOption - ステートメント・オプションの現行設定を戻す 182
SQLGetSubString - ストリング値の一部の検索 184
SQLGetTypeInfo - データ・タイプ情報の入手. 187
SQLLanguages - SQL 言語または準拠情報の取得 192
SQLMoreResults - さらに結果セットがあるかどうかの判別. 194
SQLNativeSql - 固有の SQL テキストの入手 196
SQLNextResult - 次の結果セットの処理 199
SQLNumParams - SQL ステートメント内のパラメーター数の入手 201
SQLNumResultCols - 結果列の数の取得 203
SQLParamData - データ値が必要な次のパラメーターの取得 205
SQLParamOptions - パラメーターの入力配列の指定. 207
SQLPrepare - ステートメントの準備作成 209
SQLPrimaryKeys - 表の基本キー列の入手 213
SQLProcedureColumns - プロシージャーの入出力パラメーター情報の入手 216
SQLProcedures - プロシージャー名リストの入手. 222
SQLPutData - パラメーターのデータ値に引き渡し 226
SQLReleaseEnv - すべての環境リソースの解放 229
SQLRowCount - 行数の取得 231
SQLSetConnectAttr - 接続属性の設定 233
SQLSetConnectOption - 接続オプションの設定 238

SQLSetCursorName - カーソル名の設定 240
SQLSetDescField - 記述子フィールドの設定 242
SQLSetDescRec - 記述子レコードの設定. 244
SQLSetEnvAttr - 環境属性の設定 246
SQLSetParam - パラメーターの設定 251
SQLSetStmtAttr - ステートメント属性の設定 252
SQLSetStmtOption - ステートメント・オプションの設定. 256
SQLSpecialColumns - 特殊な列 (行 ID) の取得 258
SQLStatistics - 基本表の索引情報と統計情報の取得. 262
SQLTablePrivileges - 表に関連した特権の入手. 265
SQLTables - 表情報の取得 268
SQLTransact - トランザクション管理 271

付録 A. DB2 UDB CLI 一般診断情報 273

付録 B. DB2 UDB CLI のインクルード・ファイル 275

付録 C. DB2 UDB CLI のアプリケーション・コード・リストの例 293
例: 組み込み SQL とそれと同等の DB2 UDB CLI 関数呼び出し 293
例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し 296

付録 D. サーバー・モードでの DB2 UDB CLI の実行 303
DB2 UDB CLI を SQL サーバー・モードで実行する理由 303
SQL サーバー・モードでの DB2 UDB CLI の始動 303
サーバー・モードでの DB2 UDB CLI の実行の制約事項 304

索引 305

DB2 Universal Database for iSeries SQL 呼び出しレベル・インターフェース (ODBC) について

本書は、典型的な DB2 UDB CLI アプリケーションの概要を述べています。本書には次のような内容が記載されています。

- DB2 UDB CLI を紹介し、インターフェースの背景、および組み込み SQL との関係について説明します。
- DB2 UDB CLI アプリケーション内のタスクまたはステップについて説明するとともに、概念、関数、およびそれらの相互作用について紹介します。
- DB2 UDB CLI を構成している関数を解説します。
- 次のような付録があります。
  - 273 ページの『付録 A. DB2 UDB CLI 一般診断情報』。本書全体を通して参照される表が記載されています。
  - 275 ページの『付録 B. DB2 UDB CLI のインクルード・ファイル』。すべての DB2 UDB CLI アプリケーションに組み込まれるヘッダー・ファイルを示します。
  - 293 ページの『付録 C. DB2 UDB CLI のアプリケーション・コード・リストの例』。本書全体を通して使用されるサンプル・コード・セグメントの完全なソース・コードを示します。
  - 303 ページの『付録 D. サーバー・モードでの DB2 UDB CLI の実行』。複数のユーザーが利用できるように CLI アプリケーションを使用する方法の解説が入っています。

本書の詳細については、以下のトピックを参照してください。

- 『DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) の対象読者』
- 『DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) V5R2 の新機能』
- viii ページの『コードの特記事項情報』

次に、作業を開始するには、1 ページの『第 1 章 CLI の紹介』を参照してください。

DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) の対象読者

本書は、SQL と C プログラム言語の知識があり、 DB2 UDB CLI 関数を使って動的 SQL ステートメントを呼び出すアプリケーション・プログラマーを対象とします。

DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) V5R2 の新機能

このリリースでは、以下の API が追加されました。

- SQLColumnPrivileges - 表の列に関連した特権の入手
- SQLNextResult - 次の結果セットの処理
- SQLTablePrivileges - 表に関連した特権の入手

このリリースでは、以下の API が更新されました。

- SQLBindParam - パラメーター・マーカーに対するバッファーのバインド

- SQLColAttributes - 列属性
- SQLEndTran - トランザクションのコミットまたはロールバック
- SQLGetConnectOption - 接続オプションの現行設定を戻す
- SQLGetInfo - 一般情報の取得
- SQLGetLength - ストリング値の長さの検索
- SQLGetStmtOption - ステートメント・オプションの現行設定を戻す
- SQLProcedureColumns - プロシージャーの入出力パラメーター情報の入手
- SQLProcedures - プロシージャー名リストの入手
- SQLSetConnectAttr - 接続属性の設定
- SQLSetDescRec - 記述子レコードの設定
- SQLSetEnvAttr - 環境属性の設定
- SQLSetStmtAttr - ステートメント属性の設定
- SQLTables - 表情報の取得

『CLI の紹介』と『DB2® CLI アプリケーションの作成』の情報も更新されています。

コードの特記事項情報

本書には、プログラミング・サンプルが含まれています。

IBM® は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、 IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。したがって IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用もいっさいありません。

第 1 章 CLI の紹介

DB2 UDB 呼び出しレベル・インターフェース (CLI) とは、 DB2 UDB for z/OS and OS/390® および DB2 Server for VSE and VM 以外のすべての DB2 環境でサポートされている、呼び出し可能な構造化照会言語 (SQL) プログラミング・インターフェースです。呼び出し可能 SQL インターフェースは、データベース・アクセス用の WinSock アプリケーション・プログラム・インターフェース (API) であり、動的 SQL ステートメントを始動するのに関数呼び出しを使用します。

組み込み動的 SQL の代わりに DB2 UDB CLI を使用することができます。組み込み動的 SQL と DB2 UDB CLI との間の大きな違いは、 SQL ステートメントを開始する方法にあります。 iSeries では、このインターフェースはどの ILE 言語でも利用可能です。

DB2 UDB CLI はまた、Microsoft® の ODBC のレベル 1 を全面的にサポートするとともに、レベル 2 の機能を多数提供します。 ODBC は、ほとんどの部分で ANS と ISO の SQL CLI 標準のスーパーセットとなっています。

詳細については、以下を参照してください。

- 『DB2 UDB CLI の背景情報』
- 『呼び出しレベル・インターフェース』
- 4 ページの『DB2 UDB CLI と組み込み SQL との相違』

DB2 UDB CLI の背景情報

DB2 UDB CLI またはその他の呼び出し可能な SQL インターフェースが何に基づいているかを理解し、既存のインターフェースと比較することが重要です。

ISO 標準 9075:1999 - Database Language SQL Part 3: Call-Level Interface は、CLI の標準的な定義を示しています。このインターフェースの目標は、どのデータベース・サーバーにもアプリケーションを依存させないようにすることで、アプリケーションの可搬性を高めることにあります。

ODBC は、Windows® 用のドライバー・マネージャーを備えていますが、これは、各 ODBC ドライバー (ODBC 関数呼び出しを実装していて、特定の DBMS と対話するダイナミック・リンク・ライブラリー (DLL)) のための中央制御点として働きます。

呼び出しレベル・インターフェース

iSeries でのデータベース・アクセスでは、次のような呼び出しレベル・インターフェース API を使うことができます。

- **接続**
  - 75 ページの『SQLConnect - データ・ソースへの接続』
  - 79 ページの『SQLDataSources - データ・ソース・リストの入手』
  - 90 ページの『SQLDisconnect - データ・ソースからの切断』
  - 92 ページの『SQLDriverConnect - (拡張) データ・ソースへの接続』
- **診断**
  - 98 ページの『SQLError - エラー情報の検索』
  - 149 ページの『SQLGetDiagField - 診断情報 (拡張可能) を戻す』

– 152 ページの『SQLGetDiagRec - 診断情報 (短縮型) を戻す』

- **MetaData**
  - 71 ページの『SQLColumns - 表の列情報の入手』
  - 68 ページの『SQLColumnPrivileges - 表の列に関連した特権の入手』
  - 116 ページの『SQLForeignKeys - 外部キー列リストの入手』
  - 159 ページの『SQLGetInfo - 一般情報の取得』
  - 187 ページの『SQLGetTypeInfo - データ・タイプ情報の入手』
  - 192 ページの『SQLLanguages - SQL 言語または準拠情報の取得』
  - 213 ページの『SQLPrimaryKeys - 表の基本キー列の入手』
  - 216 ページの『SQLProcedureColumns - プロシージャーの入出力パラメーター情報の入手』
  - 222 ページの『SQLProcedures - プロシージャー名リストの入手』
  - 258 ページの『SQLSpecialColumns - 特殊な列 (行 ID) の取得』
  - 262 ページの『SQLStatistics - 基本表の索引情報と統計情報の取得』
  - 265 ページの『SQLTablePrivileges - 表に関連した特権の入手』
  - 268 ページの『SQLTables - 表情報の取得』
- **SQL ステートメントの処理**
  - 61 ページの『SQLCancel - ステートメントの取り消し』
  - 62 ページの『SQLCloseCursor - カーソル・ステートメントのクローズ』
  - 63 ページの『SQLColAttributes - 列属性』
  - 83 ページの『SQLDescribeCol - 列属性の記述』
  - 87 ページの『SQLDescribeParam - パラメーター・マーカーの記述を戻す』
  - 96 ページの『SQLEndTran - トランザクションのコミットまたはロールバック』
  - 101 ページの『SQLExecDirect - ステートメントの直接実行』
  - 103 ページの『SQLExecute - ステートメントの実行』
  - 105 ページの『SQLExtendedFetch - 行配列の取り出し』
  - 108 ページの『SQLFetch - 次のデータ行』
  - 114 ページの『SQLFetchScroll - スクロール可能カーソルからの取り出し』
  - 139 ページの『SQLGetCursorName - カーソル名の取得』
  - 143 ページの『SQLGetData - 列のデータの取得』
  - 144 ページの『SQLGetDescField - 記述子フィールドの取得』
  - 147 ページの『SQLGetDescRec - 記述子レコードの取得』
  - 194 ページの『SQLMoreResults - さらに結果セットがあるかどうかの判別』
  - 196 ページの『SQLNativeSql - 固有の SQL テキストの入手』
  - 199 ページの『SQLNextResult - 次の結果セットの処理』
  - 201 ページの『SQLNumParams - SQL ステートメント内のパラメーター数の入手』
  - 203 ページの『SQLNumResultCols - 結果列の数の取得』
  - 205 ページの『SQLParamData - データ値が必要な次のパラメーターの取得』
  - 207 ページの『SQLParamOptions - パラメーターの入力配列の指定』
  - 209 ページの『SQLPrepare - ステートメントの準備作成』
  - 226 ページの『SQLPutData - パラメーターのデータ値に引き渡し』

- 231 ページの『SQLRowCount - 行数の取得』
- 240 ページの『SQLSetCursorName - カーソル名の設定』
- 271 ページの『SQLTransact - トランザクション管理』

• 属性の処理

- 130 ページの『SQLGetCol - 結果セットの行での 1 つの列の検索』
- 136 ページの『SQLGetConnectAttr - 接続属性の値の取得』
- 138 ページの『SQLGetConnectOption - 接続オプションの現行設定を戻す』
- 139 ページの『SQLGetCursorName - カーソル名の取得』
- 143 ページの『SQLGetData - 列のデータの取得』
- 144 ページの『SQLGetDescField - 記述子フィールドの取得』
- 147 ページの『SQLGetDescRec - 記述子レコードの取得』
- 155 ページの『SQLGetEnvAttr - 環境属性の現行設定を戻す』
- 156 ページの『SQLGetFunctions - 関数の取得』
- 159 ページの『SQLGetInfo - 一般情報の取得』
- 174 ページの『SQLGetLength - ストリング値の長さの検索』
- 176 ページの『SQLGetPosition - ストリングの開始位置を戻す』
- 179 ページの『SQLGetStmtAttr - ステートメント属性の値の取得』
- 182 ページの『SQLGetStmtOption - ステートメント・オプションの現行設定を戻す』
- 184 ページの『SQLGetSubString - ストリング値の一部の検索』
- 187 ページの『SQLGetTypeInfo - データ・タイプ情報の入手』
- 233 ページの『SQLSetConnectAttr - 接続属性の設定』
- 238 ページの『SQLSetConnectOption - 接続オプションの設定』
- 240 ページの『SQLSetCursorName - カーソル名の設定』
- 242 ページの『SQLSetDescField - 記述子フィールドの設定』
- 244 ページの『SQLSetDescRec - 記述子レコードの設定』
- 246 ページの『SQLSetEnvAttr - 環境属性の設定』
- 251 ページの『SQLSetParam - パラメーターの設定』
- 252 ページの『SQLSetStmtAttr - ステートメント属性の設定』
- 256 ページの『SQLSetStmtOption - ステートメント・オプションの設定』

• ハンドルの処理

- 27 ページの『SQLAllocConnect - 接続の割り振り』
- 30 ページの『SQLAllocEnv - 環境ハンドルの割り振り』
- 33 ページの『SQLAllocHandle - ハンドルの割り振り』
- 35 ページの『SQLAllocStmt - ステートメント・ハンドルの割り振り』
- 78 ページの『SQLCopyDesc - 記述ステートメントのコピー』
- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』
- 123 ページの『SQLFreeEnv - 環境ハンドルの解放』
- 125 ページの『SQLFreeHandle - ハンドルの解放』
- 127 ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』
- 229 ページの『SQLReleaseEnv - すべての環境リソースの解放』

DB2 UDB CLI と組み込み SQL との相違

組み込み SQL を使用するアプリケーションは、 SQL ステートメントをコードに変換するプリコンパイラーを必要とします。そのコードはコンパイルされ、データベースにバインドされ、実行されます。それに対して、DB2 UDB CLI アプリケーションは、プリコンパイルもバインドも必要としませんが、その代わりに、実行時に SQL ステートメントを実行して関連サービスを行うのに、標準セットの関数を使用します。

この相違は重要です。というのは、従来、プリコンパイラーはあるデータベース製品に特有のものであり、ユーザーのアプリケーションを効率よくその製品に結び付けるものであったからです。 DB2 UDB CLI を使用すると、どのデータベース製品にも限定されない可搬性のあるアプリケーションを作成することができます。製品が限定されないため、別のデータベース製品にアクセスするときでも、 DB2 UDB CLI アプリケーションを再コンパイルしたり再バインドしたりする必要はありません。アプリケーションは、実行時に適切なデータベース製品を選ぶことができます。

さらに DB2 UDB CLI と組み込み SQL とは、次の点で異なります。

- DB2 UDB CLI はカーソルの明示宣言を必要としません。 DB2 UDB CLI は必要に応じてカーソルを生成します。次いでアプリケーションは、その生成されたカーソルを、通常のカーソル取り出しモデルに従って、複数行の `SELECT` ステートメント、および位置の決まった `UPDATE` および `DELETE` ステートメント用に使用することができます。
- DB2 UDB CLI では、`OPEN` ステートメントは必要ありません。その代わり、`SELECT` の実行によってカーソルが自動的にオープンします。
- 組み込み SQL とは異なり、 DB2 UDB CLI では、 `EXECUTE IMMEDIATE` ステートメントと等価な関数 (`SQLExecDirect()` 関数) にパラメーター・マーカーを使用できます。
- DB2 UDB CLI の場合、`COMMIT` または `ROLLBACK` は、 SQL ステートメントとして受け渡されるのではなく、 `SQLTransact()` または `SQLEndTran()` 関数を通して発行されます。
- DB2 UDB CLI はアプリケーションの代わりにステートメント関連情報を管理し、その情報を抽象オブジェクトとして参照するための***ステートメント・ハンドル*** を提供します。アプリケーションは、このハンドルを使えば、製品固有のデータ構造を使用する必要がなくなります。
- ステートメント・ハンドルと同様に、***環境ハンドル*** および***接続ハンドル*** は、すべてのグローバル変数、および接続固有の情報を参照するための手段となります。
- DB2 UDB CLI は、 X/Open SQL CAE 仕様によって定義されている SQLSTATE 値を使用します。そのフォーマットおよび値の多くは、 IBM のリレーショナル・データベース製品で使用される値と一貫性がありますが、違う点もあります。

このような違いがあっても、組み込み SQL と DB2 UDB CLI に共通する重要な概念があります。

> DB2 UDB CLI は、組み込み SQL で動的に作成できる SQL ステートメントならどれでも実行できます。それは確実です。というのは DB2 UDB CLI は、 SQL ステートメントそのものを実際に*実行* するのではなく、動的に実行させるため DBMS に引き渡すからです。

表 1 は、各 SQL ステートメントと、それが DB2 UDB CLI を使用して実行できるかどうかを示しています。

表 1. SQL ステートメント

SQL ステートメント	Dyn [a]	CLI [c]
ALTER TABLE	X	X
BEGIN DECLARE SECTION [b]		
CALL	X	X

表 1. SQL ステートメント (続き)

SQL ステートメント	Dyn [a]	CLI [c]
CLOSE		SQLFreeStmt()
COMMENT ON	X	X
COMMIT	X	SQLTransact(), SQLEndTran()
CONNECT (タイプ 1)		SQLConnect()
CONNECT (タイプ 2)		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()
EXECUTE IMMEDIATE		SQLExecDirect()
FETCH		SQLFetch()
GRANT	X	X
INCLUDE [b]		
INSERT	X	X
LOCK TABLE	X	X
OPEN		SQLExecute(), SQLExecDirect()
PREPARE		SQLPrepare()
RELEASE		SQLDisconnect()
REVOKE	X	X
ROLLBACK	X	SQLTransact(), SQLEndTran()
SELECT	X	X
SET CONNECTION		
UPDATE	X	X
WHENEVER [b]		

注:

[a] **Dyn** は動的を表します。このリストにあるすべてのステートメントは静的 SQL としてコーディングできますが、 **X** のマークが付いているものは動的 SQL としてコーディングできます。

[b] これは非実行ステートメントです。

[c] X は、 SQLExecDirect() または SQLPrepare() のいずれかと、 SQLExecute() を使用してこのステートメントを実行できるということを示しています。等価の DB2 UDB CLI 関数がある場合は、関数名が示されています。

DBMS がそれぞれ、動的に作成可能な追加ステートメントを持っている場合もあります。その場合、DB2 UDB CLI はそのステートメントを DBMS へ受け渡します。 しかし例外が 1 つあります。ある種の

DBMS は COMMIT と ROLLBACK を動的に作成できますが、これらのステートメントの受け渡しは行われません。その代わりに、 SQLTransact() または SQLEndTran() を使用して、 COMMIT または ROLLBACK を指定する必要があります。

追加情報については、以下を参照してください。

- 『組み込み SQL の代わりに DB2 UDB CLI を使用する利点』
- 『DB2 UDB CLI、動的 SQL、および静的 SQL のどれがよいか』

組み込み SQL の代わりに DB2 UDB CLI を使用する利点

DB2 UDB CLI には、組み込み SQL と比べていくつかの主要な利点があります。

- DB2 UDB CLI は、クライアント/サーバー環境 (アプリケーションの構築時にはターゲット・データベースが分からない) にたいへん適しています。アプリケーションがどのデータベース・サーバーに接続されていても、 DB2 UDB CLI は、 SQL ステートメントを実行するための一貫したインターフェースを提供します。
- DB2 UDB CLI では、プリコンパイラーに依存する必要がないので、アプリケーションの可搬性が向上します。アプリケーションは、コンパイル済みのアプリケーションまたは実行時ライブラリーとしてではなく、各データベース製品用のプリプロセス済みのソース・コードとして配布されます。
- DB2 UDB CLI アプリケーションは、接続先の各データベースにバインドされる必要がありません。
- DB2 UDB CLI アプリケーションは、複数のデータベースに同時に接続することができます。
- DB2 UDB CLI アプリケーションは、組み込み SQL アプリケーションの場合のように、 SQLCA や SQLDA などのグローバル・データ域を管理する責任を負いません。その代わりに DB2 UDB CLI が必要なデータ構造を割り振って管理し、アプリケーションがそのデータ構造を参照できるようハンドル を提供します。

DB2 UDB CLI、動的 SQL、および静的 SQL のどれがよいか

どのインターフェースを選択するかは、ユーザーのアプリケーションによって異なります。

可搬性を必要とする一方で、特定の DBMS が提供する API またはユーティリティー (データベースのカタログ、バックアップ、復元など) を必要としない照会ベースのアプリケーションには、 DB2 UDB CLI が適しています。これは、DB2 UDB CLI を使用すると、アプリケーションから DBMS 固有の API が呼び出されるということではなく、アプリケーションは可搬性である必要がなくなるという意味です。

別の重要な考慮事項に、動的 SQL と静的 SQL とのパフォーマンスの比較があります。動的 SQL は実行時に作成されますが、静的 SQL は、プリコンパイルの段階で作成されます。ステートメントを作成すると、処理時間が余分に必要になるため、静的 SQL の方がより効率的といえます。動的 SQL ではなく静的 SQL を選択した場合は、DB2 UDB CLI を選ぶことはできません。

多くの場合、どちらのインターフェースを採るかは、個人の好みにゆだねられます。これまでの経験によっては、一方の方法がもう一方よりも直観的によく見えるということもあります。

第 2 章 DB2 UDB CLI アプリケーションの作成

DB2 UDB CLI アプリケーションは、一連のタスクから成りますが、さらにそれぞれのタスクは、一連の個別ステップで構成されます。アプリケーションの実行中のどこかで、他のタスクが発生することもあります。アプリケーションは 1 つ以上の DB2 UDB CLI 関数を呼び出して、そのようなタスクを 1 つずつ実行します。

どの DB2 UDB CLI アプリケーション内のタスクも、 図 1 に示されているように大きく 3 つに分かれます。 この図に示されている順序で関数が呼び出されないと、エラーが生じます。

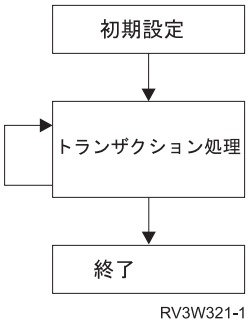


図 1. DB2 UDB CLI アプリケーションの概念図

初期設定
このタスクは、主要なタスクである*トランザクション処理* の準備段階で、いくつかのリソースを割り振り、初期設定を行います。詳細については、 8 ページの『DB2 UDB CLI アプリケーションでの初期設定と終了のタスク』を参照してください。

トランザクション処理
これはアプリケーションの主要なタスクです。 SQL ステートメントは、 SQL の照会と変更のために DB2 UDB CLI に引き渡されます。 詳細については、 10 ページの『DB2 UDB CLI アプリケーションでのトランザクション処理』を参照してください。

終了 このタスクは、割り振られたリソースを解放します。一般にリソースは、固有のハンドルで識別されるデータ領域から構成されます。リソースの解放が終わると、その他のタスクがそのハンドルを使用できるようになります。詳細については、 8 ページの『DB2 UDB CLI アプリケーションでの初期設定と終了のタスク』を参照してください。

上に示されている 3 つのタスクに加えて、処理診断メッセージのような、アプリケーション全体のどこかで生じる一般的なタスクもあります。

このトピックでは、これらの関数を DB2 UDB CLI アプリケーションで使用する方法を説明するための例が用意されています。

追加情報については、以下を参照してください。

- 8 ページの『DB2 UDB CLI アプリケーションでの初期設定と終了のタスク』
- 10 ページの『DB2 UDB CLI アプリケーションでのトランザクション処理』
- 16 ページの『DB2 UDB CLI アプリケーションでの診断』
- 17 ページの『DB2 UDB CLI の関数でのデータ・タイプとデータ変換』
- 20 ページの『DB2 UDB CLI 関数でのストリング引き数の処理』

それぞれの関数の詳しい説明と使用法については、 23 ページの『第 3 章 DB2 UDB CLI の関数』を参照してください。

DB2 UDB CLI アプリケーションでの初期設定と終了のタスク

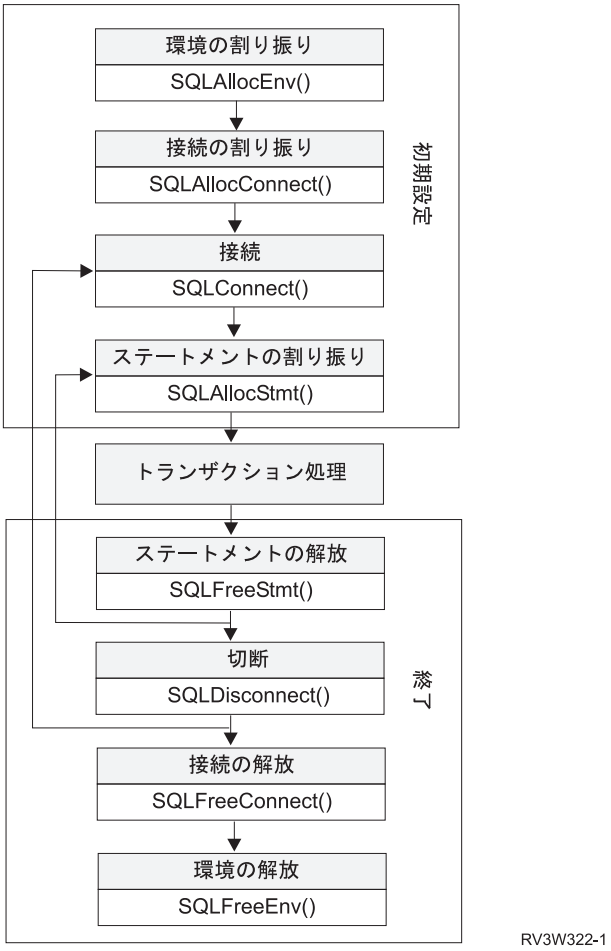


図 2. 初期設定および終了タスクの概念図

図 2 は、初期設定タスクと終了タスクの関数呼び出しの順序を示しています。図の中央にあるトランザクション処理タスクは、 11 ページの図 3 に示されています。

初期設定タスクは、環境ハンドルと接続ハンドルを割り振って初期設定します。終了タスクはそれらを解放します。ハンドルとは、 DB2 UDB CLI によって制御されるデータ・オブジェクトを参照する変数です。アプリケーションは、ハンドルを使用すると、グローバル変数またはデータ構造 (たとえば、 IBM DBMS 用の組み込み SQL インターフェースで使用される SQLDA または SQLCA など) の割り振りと管理を行う必要がなくなります。その後、アプリケーションは、その他の DB2 UDB CLI 関数を呼び出すときに、該当するハンドルを受け渡します。 ハンドルには、次の 3 つのタイプがあります。

環境ハンドル

環境ハンドルは、アプリケーションの状態に関するグローバルな情報の入ったデータ・オブジェクトを参照します。このハンドルは、 SQLAllocEnv() の呼び出しで割り振られ、 SQLFreeEnv() の呼び出しで解放されます。接続ハンドルを割り振るには、あらかじめ環境ハンドルを割り振っておく必要があります。アプリケーションごとに環境ハンドルを 1 つだけ割り振ることができます。

接続ハンドル

接続ハンドルは、 DB2 UDB CLI によって管理される接続に関連した情報の入ったデータ・オブジェクトを参照します。 これには一般状況情報、トランザクション状況、および診断情報が含まれます。各接続ハンドルは、 SQLAllocConnect() の呼び出しで割り振られ、 SQLFreeConnect() の呼び出しで解放されます。アプリケーションは、データベース・サーバーへの接続ごとに接続ハンドルを 1 つずつ割り振る必要があります。

ステートメント・ハンドル

ステートメント・ハンドルについては次のタスクの中で説明します。

『例: DB2 UDB CLI アプリケーションでの初期設定と接続』 を参照してください。

例: DB2 UDB CLI アプリケーションでの初期設定と接続

コード例については、 viii ページの『コードの特記事項情報』を参照してください。

```
/*************************************************
** 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 アプリケーションでのトランザクション処理

次の図は、 DB2 UDB CLI アプリケーションでの典型的な関数呼び出しの順序を示しています。すべての関数または使用可能なパスが示されているわけではありません。

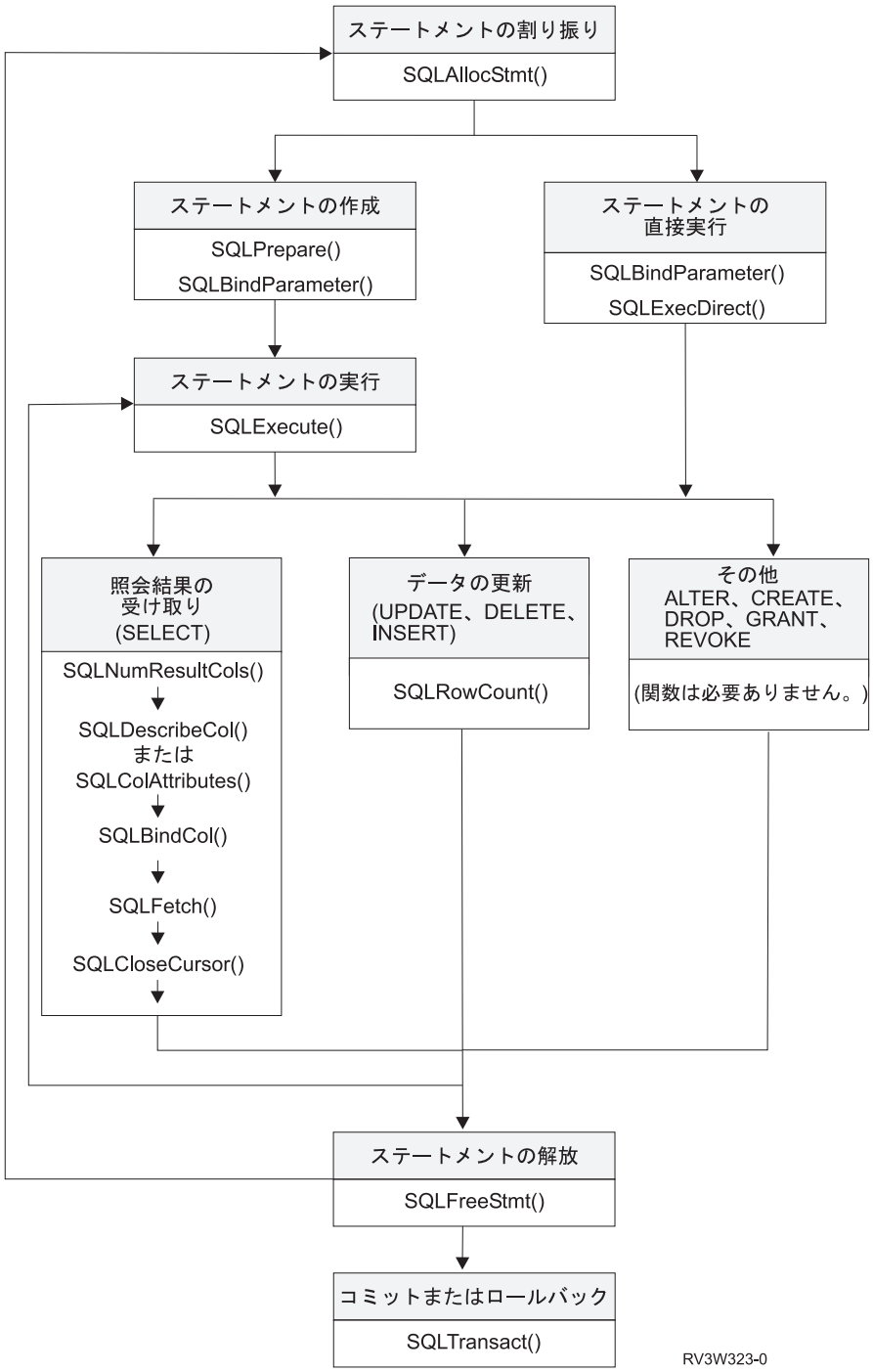


図 3. トランザクション処理

図 3 は、トランザクション処理タスク内のステップおよび DB2 UDB CLI の関数を示しています。このタスクには、次のような 5 つのステップが入っています。

- 12 ページの『DB2 UDB CLI アプリケーションでのステートメント・ハンドルの割り振り』
- 12 ページの『DB2 UDB CLI アプリケーションでの作成タスクと実行タスク』
- 13 ページの『DB2 UDB CLI アプリケーションでの処理結果』
- 15 ページの『DB2 UDB CLI アプリケーションでのステートメント・ハンドルの解放』
- 15 ページの『DB2 UDB CLI アプリケーションでのコミットまたはロールバック』

関数 SQLAllocStmt は、SQL ステートメントの処理に使用されるステートメント・ハンドルを入手するために必要になります。ステートメントを実行するために使用できる方式は 2 つあります。 SQLPrepare と SQLExecute を使用すると、プログラムはプロセスを 2 つのステップに分割できます。関数 SQLBindParameter は、プログラムのアドレスを、作成される SQL ステートメントで使用されるホスト変数にバインドするために使用されます。 2 番目の方式は直接実行方式です。この方式では、SQLPrepare と SQLExecute が SQLExecDirect の単一の呼び出しで置き換えられます。

ステートメントを実行した後の残りの処理は、SQL ステートメントのタイプによって異なります。SELECT ステートメントの場合、プログラムは SQLNumResultCols、SQLDescribeCol、SQLBindCol、SQLFetch、および SQLCloseCursor などの関数を使用して、結果セットを処理します。データを更新するステートメントの場合は、影響を受ける行数を判別するために SQLRowCount を使用できるかもしれません。他のタイプの SQL ステートメントの場合、処理はステートメントが実行された後に完了します。すべての場合において、ハンドルが必要なくなったことを示すため、その後に SQLFreeStmt が使用されます。

DB2 UDB CLI アプリケーションでのステートメント・ハンドルの割り振り

SQLAllocStmt() はステートメント・ハンドルを割り振ります。ステートメント・ハンドルは、 DB2 UDB CLI によって管理される SQL ステートメントに関する情報の入ったデータ・オブジェクトを参照します。これには、動的引き数、カーソル情報、動的引き数と列のバインド、結果値および状況情報 (これらについては後で述べます) のような情報が含まれます。各ステートメント・ハンドルは接続ハンドルと関連しています。

ステートメント・ハンドルを割り振って、ステートメントを実行します。同時に割り振れるハンドルの合計数は 80,000 に制限されています。この制限は、実装コードによって暗黙で割り振られる記述子ハンドルを含め、すべてのタイプのハンドルに適用されます。 また、リモート接続の場合は、ステートメント・ハンドル数が 500 に制限されています。

DB2 UDB CLI アプリケーションでの作成タスクと実行タスク

ステートメント・ハンドルの割り振りが済んだ後、 SQL ステートメントを指定して実行するには次の 2 通りの方法があります。

1. 次のように準備してから実行します。
   a. 引き数として SQL ステートメントを指定して、 SQLPrepare() を呼び出す。
   b. SQL ステートメントにパラメーター・マーカー が入っている場合は、 SQLSetParam() を呼び出す。
   c. SQLExecute() を呼び出す。
2. 次のように直接実行します。
   a. SQL ステートメントにパラメーター・マーカー が入っている場合は、 SQLSetParam() を呼び出す。
   b. 引き数として SQL ステートメントを用いて、 SQLExecDirect() を呼び出す。

1 番目の方式は、ステートメントの作成と実行とを分離しています。この方式は、次の場合に使用されます。

- ステートメントが (通常は異なるパラメーター値で) 繰り返し実行される場合。こうすると、同じステートメントを 2 回以上作成する必要がなくなります。
- ステートメント実行の前に、アプリケーションが結果セット内の列についての情報を必要とする場合。

2 番目の方式は、作成ステップと実行ステップを 1 つにまとめています。この方式は、次の場合に使用されます。

- ステートメントが一度だけ実行される場合。こうすれば、ステートメントを実行するのに 2 つの関数を呼び出さずに済みます。
- ステートメント実行の前に、アプリケーションが結果セット内の列についての情報を必要としない場合。

DB2 UDB CLI アプリケーションでの SQL ステートメント内のパラメーターのバインド

この 2 つのどちらの実行方式でも、 SQL ステートメント内の式 (または組み込み SQL 内のホスト変数) の代わりに、パラメーター・マーカーを使用することができます。

パラメーター・マーカーは '?' 文字で表され、 SQL ステートメントの実行時にアプリケーション変数の内容が置換される SQL ステートメント内の位置を示します。マーカーは、1 から始まって、左から右へ順番に参照されます。

アプリケーション変数がパラメーター・マーカーと関連付けられると、それはパラメーター・マーカーにバインド されます。バインドは、次のものを指定した SQLSetParam() 関数の呼び出しで実行されます。

- パラメーター・マーカーの数
- アプリケーション変数を指すポインター
- パラメーターの SQL タイプ
- 変数のデータ・タイプと長さ

SQLSetParam() の呼び出し時にポインターだけが引き渡されるため、このようなアプリケーション変数を、*据え置き* 引き数と呼びます。ステートメントが実行されない限り、変数からデータは読み込まれません。これは、バッファー引き数に対してと、バッファー内のデータの長さを示す引き数に対して適用されます。アプリケーションは、据え置き引き数を使うと、バインドされたパラメーター変数の内容を変更してから、新規の値を使ってステートメントの実行を繰り返すことができます。

SQLSetParam() の呼び出し時に、 SQL ステートメントの必須タイプのものとは異なるタイプの変数をバインドすることができます。この場合、DB2 UDB CLI はバインドされた変数を正しいタイプのものへ変換します。 たとえば、整数値を必要とする SQL ステートメントの場合に、アプリケーションは整数のストリング表記を持っているとします。そのストリングをパラメーターにバインドして、ステートメントの実行時に DB2 UDB CLI でそのストリングを整数に変換することができます。データ変換の詳細については、17 ページの『DB2 UDB CLI の関数でのデータ・タイプとデータ変換』を参照してください。

詳細および例については、次を参照してください。

- 209 ページの『SQLPrepare - ステートメントの準備作成』
- 251 ページの『SQLSetParam - パラメーターの設定』
- 103 ページの『SQLExecute - ステートメントの実行』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』

DB2 UDB CLI アプリケーションでの処理結果

ステートメントの実行後の次のステップは、 SQL ステートメントのタイプによって異なります。

DB2 UDB CLI アプリケーションでの SELECT ステートメントの処理

ステートメントが SELECT の場合、結果セットの各行を検索するには、一般に次のステップが必要です。

1. 結果セットの構造、列の数、列のタイプおよび長さを確立します。
2. データを受け取るため、アプリケーション変数を列にバインドします (オプション)。

3. 次の行のデータを繰り返し取り出し、それをバインドされたアプリケーション変数の中で受け取ります。
4. 以前にバインドされていない列は、正常に実行されたそれぞれの取り出しの後、 SQLGetData() の呼び出しで検索できます (オプション)。

注: 上のステップのいずれにおいても、いくつかの診断チェックが必要です。

1 番目のステップでは、実行または作成されたステートメントの分析が必要です。 SQL ステートメントがアプリケーションによって生成されたものである場合、このステップは必要ありません。なぜなら、結果セットの構造および各列のデータ・タイプはアプリケーションで分かっているからです。 SQL ステートメントが (たとえば、ユーザーの入力によって) 実行時に生成されたものである場合、アプリケーションは次のものを照会する必要があります。

- 列の数
- 各列のタイプ
- 結果セット内の各列の名前

この情報は、ステートメントの作成後またはステートメントの実行後、 SQLNumResultCols() および SQLDescribeCol() (または SQLColAttributes()) の呼び出しで取得できます。

2 番目のステップは、アプリケーションが次の SQLFetch() 呼び出しで、列データをアプリケーション変数に直接取り込めるようにします。取り出される各列ごとにアプリケーションは SQLBindCol() を呼び出して、アプリケーション変数を結果セット内の列にバインドします。 SQLSetParam() を使用して変数をパラメーター・マーカーにバインドするのと同様に、据え置き引き数を使用して列がバインドされます。この場合は変数が出力引き数であり、SQLFetch() が呼び出されるときに、データがそれらの変数に書き込まれます。 SQLGetData() を使用してデータを取り出すこともできるため、 SQLBindCol() の呼び出しはオプションです。

3 番目のステップは、SQLFetch() を呼び出して、結果セットの 1 番目またはその次の行を取り出します。いずれかの列がバインドされている場合は、アプリケーション変数が更新されます。 SQLBindCol の呼び出しに指定されているデータ・タイプが何らかのデータ変換の必要性を示している場合、 SQLFetch() が呼び出されるときにその変換が生じます。データ変換については、 17 ページの『DB2 UDB CLI の関数でのデータ・タイプとデータ変換』を参照してください。

最後の (オプションの) ステップでは、 SQLGetData() を呼び出して、以前にバインドされていない列を取り出します。列がバインドされていないかぎり、すべての列をこの方法で取り出すことができます。あるいは、両方の方法を組み合わせて取り出すこともできます。 SQLGetData() はまた、可変長列をより小さな部分に分けて取り出す場合にも有効ですが、バインドされた列を取り出すことはできません。 SQLBindCol() と同様、ここでもデータ変換を指示することができます。詳細については、 17 ページの『DB2 UDB CLI の関数でのデータ・タイプとデータ変換』を参照してください。

詳細および例については、次を参照してください。

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 63 ページの『SQLColAttributes - 列属性』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 108 ページの『SQLFetch - 次のデータ行』
- 143 ページの『SQLGetData - 列のデータの取得』
- 203 ページの『SQLNumResultCols - 結果列の数の取得』

DB2 UDB CLI アプリケーションでの UPDATE、DELETE、および INSERT ステートメントの処理

ステートメントがデータを変更するものである場合 (UPDATE、DELETE または INSERT)、診断メッセージがあるかどうかを調べる通常のチェック以外には、何の処置も必要ありません。この場合、 SQL ステートメントによって影響を受ける行の数を獲得するために、 SQLRowCount() を使用できます。詳細については、 203 ページの『SQLNumResultCols - 結果列の数の取得』を参照してください。

SQL ステートメントが、位置の決まった UPDATE または DELETE である場合は、*カーソル* を使用する必要があります。カーソルは、 SELECT ステートメントの結果表内の行を指す移動可能なポインターです。組み込み SQL の場合、行の取り出し、更新、または削除にカーソルを利用します。 DB2 UDB CLI を使用する場合、カーソルは自動的に生成されるため、定義する必要はありません。

位置の決まった UPDATE または DELETE ステートメントの場合は、 SQL ステートメントでカーソルの名前を指定する必要があります。 SQLSetCursorName() を使用して独自のカーソル名を定義することができますが、 SQLGetCursorName() を使用して、生成されたカーソルの名前を照会することもできます。すべてのエラー・メッセージは、 SQLSetCursorName() で定義された名前ではなく、生成されたカーソル名を参照するため、生成された名前を使うのが最善の方法です。

DB2 UDB CLI アプリケーションでのその他の SQL ステートメントの処理

ステートメントがデータを照会も変更もしない場合は、診断メッセージがあるかどうかを調べる通常のチェック以外に必要な処置はありません。

DB2 UDB CLI アプリケーションでのステートメント・ハンドルの解放

特定のステートメント・ハンドルの処理を終了するには、 SQLFreeStmt() を呼び出します。この関数を使って、以下の中の 1 つ以上の処理を行うことができます。

- すべての列のアンバインド
- すべてのパラメーターのアンバインド
- すべてのカーソルのクローズと結果の廃棄
- ステートメント・ハンドルのドロップ、および全関連リソースの解放

ステートメント・ハンドルは、ドロップしないかぎり再使用できます。

DB2 UDB CLI アプリケーションでのコミットまたはロールバック

最後のステップでは、SQLTransact() を使用して、*トランザクション* をコミットまたはロールバックします。

トランザクションとは、リカバリー可能な作業単位です。つまり、1 つの分割不能な操作として取り扱うことのできる SQL ステートメントのグループです。つまり、グループ内のすべての操作を単一の操作であるものとして、完了 (コミット) したり、やり直し (ロールバック) したりするということです。

DB2 UDB CLI の使用時には、SQLPrepare()、SQLExecDirect()、または SQLGetTypeInfo() を使用してデータベースに最初にアクセスすると、トランザクションが暗黙で開始されます。 トランザクションをロールバックまたはコミットする SQLTransact() を使用すると、トランザクションが終了します。これは、この 2 つの関数の間に実行される SQL ステートメントは 1 つの作業単位として処理されることを意味します。

DB2 UDB CLI アプリケーションでの SQLTransact() の呼び出し時期

トランザクションをいつ終わらせるかを決定するときは、次の事項を考慮してください。

- 現行のトランザクションは、コミットまたはロールバックするしかないので、依存しあうステートメントは、同じトランザクション内に入れておいてください。
- 未処理のトランザクションがある間、各種のロックが設定されています。トランザクションが終了するとそれらのロックが解除され、他のユーザーがデータにアクセスできるようになります。これは、SELECT ステートメントを含め、すべての SQL ステートメントにあてはまります。
- トランザクションは、コミットまたはロールバックの正常完了後に、システム・ログから全面的にリカバリーできるようになります (これは DBMS に依存します)。オープン・トランザクションはリカバリー可能ではありません。

DB2 UDB CLI アプリケーションでの SQLTransact() の呼び出しの効果

トランザクションが終了すると、次の事柄があてはまります。

- 再利用のためには、すべてのステートメントを、あらかじめ準備しておかなければなりません。
- カーソルの名前、バインドされたパラメーター、および列のバインドは、トランザクションからトランザクションへ持ち越されます。
- すべてのオープン・カーソルはクローズされます。

詳細および例については、 271 ページの『SQLTransact - トランザクション管理』を参照してください。

DB2 UDB CLI アプリケーションでの診断

診断は、アプリケーション内で生成される警告またはエラー状態の処理を示します。 DB2 UDB CLI の関数を呼び出す場合の診断には、次の 2 つのレベルがあります。

- 『DB2 UDB CLI アプリケーションでの戻りコード』
- 17 ページの『DB2 UDB CLI SQLSTATE』 (診断メッセージ)

エラー処理の例については、 98 ページの『SQLError - エラー情報の検索』を参照してください。

DB2 UDB CLI アプリケーションでの戻りコード

次の表は、DB2 UDB CLI 関数で生じうるすべての戻りコードを示しています。 23 ページの『第 3 章 DB2 UDB CLI の関数』 内にある各関数の説明の部分には、各関数で生じうる戻りコードが示されています。

表 2. DB2 UDB CLI 関数戻りコード

戻りコード	解説
SQL_SUCCESS	関数は正常に完了し、追加の SQLSTATE 情報はありません。
SQL_SUCCESS_WITH_INFO	関数は正常に完了しましたが、警告またはその他の情報があります。 SQLSTATE およびその他のエラー情報を受け取るには、 SQLError() を呼び出してください。 SQLSTATE のクラスは '01' です。
SQL_NO_DATA_FOUND	関数の戻りは正常に完了しましたが、関係データが見つかりません。
SQL_ERROR	関数は失敗しました。 SQLSTATE およびその他のエラー情報を受け取るには、 SQLError() を呼び出してください。
SQL_INVALID_HANDLE	関数は無効な入力ハンドル (環境、接続、またはステートメント・ハンドル) のために失敗しました。

DB2 UDB CLI SQLSTATE

データベース・サーバーが異なれば、診断メッセージ・コードも異なることが多いので、DB2 UDB CLI では、 X/Open SQL CAE 仕様で定義された標準セットの *SQLSTATE* が用意されています。そうすれば、データベース・サーバーが異なっても、一貫したメッセージ処理を行うことができます。

SQLSTATE は ccsss のフォーマットを持つ 5 文字 (バイト) の英数字のストリングです。 cc はクラスを指し、 sss はサブクラスを指します。 SQLSTATE については、次のことがいえます。

- クラスが '01' の場合は警告です。
- クラスが 'HY' の場合、コマンド行インターフェース (CLI) ドライバー (DB2 UDB CLI またはオープン・データベース・コネクティビティー (ODBC)) によって生成されます。

エラー・コードがサーバーによって生成された場合、 SQLError() 関数も*固有の* エラー・コードを戻します。 IBM データベース・サーバーに接続している場合、固有エラー・コードは SQLCODE になります。そのコードがサーバーではなく DB2 UDB CLI によって生成された場合には、固有エラー・コードは -99999 に設定されます。

DB2 UDB CLI の SQLSTATE には、データベース・サーバーから戻される IBM 定義の付加的な SQLSTATE と、 X/Open 仕様で定義されていない条件用の DB2 UDB CLI 定義の SQLSTATE の両方が含まれます。このようにして、最大限の量の診断情報が戻されるようになっています。 ODBC を使用して Windows でアプリケーションを実行すると、 ODBC 定義の SQLSTATE も受け取ることができます。

アプリケーション内での SQLSTATE の使用については、次の指針に従ってください。

- SQLError() を呼び出すには、その前に必ず関数戻りコードを調べ、診断情報が利用可能かどうかを判別してください。
- 固有エラー・コードよりも SQLSTATE を使用するようにしてください。
- アプリケーションの可搬性を高めるためには、X/Open 仕様によって定義されている DB2 UDB CLI の SQLSTATE のサブセットへの従属関係だけを構築するようにし、付加的なものは情報としてのみ戻すようにしてください。 (依存性は、特定の SQLSTATE に基づいて論理の流れの決定を行うアプリケーションに関係します。)
- 診断情報量を最大化するには、テキスト・メッセージを SQLSTATE と一緒に戻すようにしてください (該当する場合には、テキスト・メッセージには IBM 定義の SQLSTATE が含まれます)。また、エラーを戻した関数の名前をアプリケーションで出力しても役にたちます。

DB2 UDB CLI の関数でのデータ・タイプとデータ変換

18 ページの表 3 は、サポートされている SQL タイプとそれに対応する記号名をすべて示しています。SQLBindParam()、SQLBindParameter()、SQLSetParam()、SQLBindCol()、および SQLGetData() において、引き数のデータ・タイプを示すのに記号名が使われています。

各列について、以下に説明します。

SQL タイプ

この列には、 SQL ステートメントに現れる形で SQL データ・タイプが示されています。 SQL データ・タイプは DBMS に準じます。

SQL 記号

この列には、整数値として (sqlcli.h 内に) 定義されている SQL 記号名が入っています。この値は、 1 番目の列の SQL データ・タイプを識別するのに各種関数で使用されます。

表 3. SQL データ・タイプとデフォルト C データ・タイプ

SQL タイプ	SQL 記号
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
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、および TIMESTAMP の値は、文字形式で戻されます。	
[2] SQL_WCHAR と SQL_WVARCHAR は、 Unicode データを示すのに使うことができます。	

詳細については、以下を参照してください。

- 『DB2 UDB CLI 関数でのその他の C データ・タイプ』
- 19 ページの『DB2 UDB CLI 関数でのデータ変換』

DB2 UDB CLI 関数でのその他の C データ・タイプ

SQL データ・タイプにマップされるデータ・タイプの他に、ポインターやハンドルのように、その他の関数の引き数に使用される C 記号タイプもあります。

表 4. 総称データ・タイプと実際の C データ・タイプ

記号タイプ	実際の C タイプ	典型的な使用法
SQLPOINTER	void *	データとパラメーター用のストレージを指すポインター
SQLHENV	long int	環境情報を参照するハンドル
SQLHDBC	long int	データベース接続情報を参照するハンドル
SQLHSTMT	long int	ステートメント情報を参照するハンドル
SQLRETURN	long int	DB2 UDB CLI 関数からの戻りコード

DB2 UDB CLI 関数でのデータ変換

前に述べたように、 DB2 UDB CLI はアプリケーションと DBMS との間の転送と、必要なデータ変換を管理します。 データ転送が実際に行われる前に、 SQLBindParam()、SQLBindParameter()、SQLSetParam()、SQLBindCol() または SQLGetData() の呼び出し時に、ソースとターゲットの片方または両方のデータ・タイプが指定されます。これらの関数は、 18 ページの表 3 に示される記号タイプ名を使用して、そこに含まれているデータ・タイプを識別します。記号データ・タイプを使用する関数の例については、 109 ページの『例』 (SQLFetch())、または 133 ページの『例』 (SQLGetCol()) を参照してください。

表 5 は、 DB2 UDB CLI でサポートされている変換を示しています。 デフォルト変換のみを示しています。実行するステートメントの SQL 構文内で SQL スカラー関数または SQL CAST 関数を使えば、その他の変換も可能になります。

前の段落で述べた関数を使って、データをその他のタイプに変換することができます。すべてのデータ変換がサポートされているわけでも、それらすべてが妥当であるわけでもありません。 表 5 は、 DB2 UDB CLI でサポートされている変換を示しています。

表 5 内の 1 番目の列にはソース・データ・タイプが示されていて、残りの列にはターゲット・データ・タイプが示されています。 X は、DB2 UDB CLI がその変換をサポートすることを示します。

表 5. サポートされるデータ変換

ソース・データ・タイプ	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			

表 5. サポートされるデータ変換 (続き)

ソース・データ・タイプ	VARGRAPHIC	GRAPHIC	TIMESTAMP	TIME	DATE	VARCHAR	DOUBLE	REAL	FLOAT	SMALLINT	BIGINT	INTEGER	DECIMAL	NUMERIC	CHAR	BLOB	CLOB	DBCLOB
注:																		
1. REAL は、DB2 UDB for OS/2® または DB2 UDB for AIX/6000 ではサポート**されていません**。																		
2. NUMERIC は DB2 UDB for iSeries **のみ**サポートされています (その他の DBMS では DECIMAL として扱われます)。																		

関数呼び出しで、丸めのための切り捨て、またはデータ・タイプの非互換性が生じるたびに、SQL_ERROR または SQL_SUCCESS_WITH_INFO が返されます。次いで詳細な情報が SQLSTATE 値で示され、その他の情報が SQLError() によって戻されます。

DB2 UDB CLI 関数でのストリング引き数の処理

次のような規則が、 DB2 UDB CLI 関数内のストリング引き数の処理をさまざまな側面で規制します。

- 『DB2 UDB CLI 関数でのストリング引き数の長さ』
- 21 ページの『DB2 UDB CLI 関数でのストリングの切り捨て』
- 21 ページの『DB2 UDB CLI 関数でのストリングの解釈』

DB2 UDB CLI 関数でのストリング引き数の長さ

入力ストリング引き数には、関連した長さ引き数があります。この引き数は、割り振られたバッファーの長さ (NULL バイトの終了文字を含まない) または特殊値の SQL_NTS のいずれかを DB2 UDB CLI に示します。 SQL_NTS が渡された場合、DB2 UDB CLI は、 NULL 終了文字を見つけ出してストリングの長さを判別します。

出力ストリング引き数には、関連した長さの引き数が 2 つあります。 1 つは割り振られたバッファーの長さを指定し、もう 1 つは DB2 UDB CLI から戻されたストリングの長さを戻します。 戻される長さの値は、バッファーに入りきるかどうかに関係なく、戻すのに使用できるストリングの合計の長さになります。

SQL 列データの場合に、出力が NULL ストリングであると、 SQL_NULL_DATA が長さ引き数に戻されます。

出力長さ引き数に NULL ポインターを指定して関数が呼び出された場合、 DB2 UDB CLI は長さを戻しません。 これは、どのような結果が生じてもバッファーの大きさは十分であることが明らかな場合には、便利かもしれません。列に NULL データが入っていることを示すために DB2 UDB CLI が SQL_NULL_DATA 値を戻そうとした場合に、出力長さ引き数が NULL ポインターであると、関数呼び出しは失敗します。

グラフィック・データ・タイプから戻されるストリングを除き、 DB2 UDB CLI が戻すすべての文字ストリングは、NULL 終了文字 (16 進数 00) で終わります。 そのため、予想最大量を入れるのに十分なスペースに、 NULL 終了文字のための 1 文字分を加えたスペースを、すべてのバッファーに割り振る必要があります。

DB2 UDB CLI 関数でのストリングの切り捨て

出力ストリングがバッファーに入りきらない場合、 DB2 UDB CLI はバッファーのサイズより 1 小さい長さにストリングを切り捨ててから、 NULL 終了文字を書き込みます。 切り捨てが起きた場合に、関数は SQL_SUCCESS_WITH_INFO および切り捨てが起こったことを示す SQLSTATE を戻します。すると、アプリケーションは、バッファー長さを出力長と比較し、どのストリングが切り捨てられたかを判別することができます。

たとえば、 SQLFetch() が SQL_SUCCESS_WITH_INFO および 01004 の SQLSTATE を戻した場合、列にバインドされたバッファーのうち最低 1 つは、データを収容するには小さすぎることが分かります。アプリケーションは、列にバインドされた各バッファーごとに、そのバッファー長を出力長と比較して、どの列が切り捨てられたかを判別できます。

DB2 UDB CLI 関数でのストリングの解釈

DB2 UDB CLI では、大文字小文字は無視され、列の名前やカーソルの名前のような、すべてのストリング入力引き数の前後のブランクは除去されます。ただし、次のものは例外です。

- データベース・データ
- 二重引用符で囲まれている区切り文字付き ID
- パスワード引き数

第 3 章 DB2 UDB CLI の関数

このトピックでは、それぞれの関数について説明します。 DB2 UDB CLI の各関数には次のような情報が入っています。

- **目的**

  この項では、該当関数の機能を簡単に概説します。また、説明中の関数を呼び出す前後に呼び出す必要のある関数についても説明します。

- **構文**

  この項では、OS/400® 環境の 'C' プロトタイプについて述べます。

- **引き数**

  この項では、それぞれの関数の引き数、そのデータ・タイプ、説明、および入力引き数か出力引き数かをリストします。

  それぞれの DB2 UDB CLI 引き数は、入力引き数か出力引き数のどちらかです。 `SQLGetInfo()` を除き、出力になっている引き数のみが DB2 UDB CLI により変更されます。

  関数によっては、*据え置き* 引き数または*バインド* 引き数として知られる入力引き数または出力引き数を持つものもあります。これらの引き数は、アプリケーションで割り振られているバッファーへのポインターになっています。これらの引き数は、SQL ステートメントのパラメーター、または結果セットの列に関連 (またはバインド) しています。この関数によって指定されるデータ域は、あとで DB2 UDB CLI からアクセスされます。 このため、 DB2 UDB CLI がこれらの据え置きデータ域にアクセスするときに、これらのデータ域が有効のままであることは重要です。

- **使用法**

  この項では、該当関数を使用する方法、および特殊な考慮事項などの情報を示します。推定エラー状態についてはこの項では説明しませんが、その代わりに診断セクションでリストにして示されています。

- **戻りコード**

  この項には、有効な関数戻りコードがすべてリストされます。 SQL_ERROR または SQL_SUCCESS_WITH_INFO が戻された場合、 `SQLError()` を呼び出してエラー情報を入手することができます。

  戻りコードの詳細については、 16 ページの『DB2 UDB CLI アプリケーションでの診断』を参照してください。

- **診断**

  この項では、 DB2 UDB CLI によって明示的に戻される SQLSTATE (DBMS 生成の SQLSTATE も戻される場合があります) を表にリストし、エラーの原因を示します。 該当関数から SQL_ERROR または SQL_SUCCESS_WITH_INFO が戻された後で `SQLError()` を呼び出せば、これらの値を得られます。

  最初の列の 『*****』 は、 SQLSTATE は DB2 UDB CLI からのみ戻され、他の ODBC ドライバーからは戻されないことを表します。

  診断の詳細については、 16 ページの『DB2 UDB CLI アプリケーションでの診断』を参照してください。

- **制約事項**

  この項では、アプリケーション・プログラムに影響を与える可能性のある DB2 UDB CLI と ODBC との相違点または制限事項を記載します。

- **例**

この項は、該当関数の使用法の実例を示すコーディングの一部となっています。すべてのコーディング例を使用した完全なソースは、 293 ページの『付録 C. DB2 UDB CLI のアプリケーション・コード・リストの例』にリストされています。

- **参照**

  この項には、関連する DB2 UDB CLI 関数がリストされます。

関数は以下のとおりです。

- 27 ページの『SQLAllocConnect - 接続の割り振り』
- 30 ページの『SQLAllocEnv - 環境ハンドルの割り振り』
- 33 ページの『SQLAllocHandle - ハンドルの割り振り』
- 35 ページの『SQLAllocStmt - ステートメント・ハンドルの割り振り』
- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 42 ページの『SQLBindFileToCol - LOB 列に対する LOB ファイル参照のバインド』
- 45 ページの『SQLBindFileToParam - LOB パラメーターに対する LOB ファイル参照のバインド』
- 48 ページの『SQLBindParam - パラメーター・マーカーに対するバッファーのバインド』
- 53 ページの『SQLBindParameter - バッファーに対するパラメーター・マーカーのバインド』
- 61 ページの『SQLCancel - ステートメントの取り消し』
- 62 ページの『SQLCloseCursor - カーソル・ステートメントのクローズ』
- 63 ページの『SQLColAttributes - 列属性』
- 68 ページの『SQLColumnPrivileges - 表の列に関連した特権の入手』
- 71 ページの『SQLColumns - 表の列情報の入手』
- 75 ページの『SQLConnect - データ・ソースへの接続』
- 78 ページの『SQLCopyDesc - 記述ステートメントのコピー』
- 79 ページの『SQLDataSources - データ・ソース・リストの入手』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 87 ページの『SQLDescribeParam - パラメーター・マーカーの記述を戻す』
- 90 ページの『SQLDisconnect - データ・ソースからの切断』
- 92 ページの『SQLDriverConnect - (拡張) データ・ソースへの接続』
- 96 ページの『SQLEndTran - トランザクションのコミットまたはロールバック』
- 98 ページの『SQLError - エラー情報の検索』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 105 ページの『SQLExtendedFetch - 行配列の取り出し』
- 108 ページの『SQLFetch - 次のデータ行』
- 114 ページの『SQLFetchScroll - スクロール可能カーソルからの取り出し』
- 116 ページの『SQLForeignKeys - 外部キー列リストの入手』
- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』
- 123 ページの『SQLFreeEnv - 環境ハンドルの解放』
- 125 ページの『SQLFreeHandle - ハンドルの解放』
- 127 ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』
- 130 ページの『SQLGetCol - 結果セットの行での 1 つの列の検索』

- 136 ページの『SQLGetConnectAttr - 接続属性の値の取得』
- 138 ページの『SQLGetConnectOption - 接続オプションの現行設定を戻す』
- 139 ページの『SQLGetCursorName - カーソル名の取得』
- 143 ページの『SQLGetData - 列のデータの取得』
- 144 ページの『SQLGetDescField - 記述子フィールドの取得』
- 147 ページの『SQLGetDescRec - 記述子レコードの取得』
- 149 ページの『SQLGetDiagField - 診断情報 (拡張可能) を戻す』
- 152 ページの『SQLGetDiagRec - 診断情報 (短縮型) を戻す』
- 155 ページの『SQLGetEnvAttr - 環境属性の現行設定を戻す』
- 156 ページの『SQLGetFunctions - 関数の取得』
- 159 ページの『SQLGetInfo - 一般情報の取得』
- 174 ページの『SQLGetLength - ストリング値の長さの検索』
- 176 ページの『SQLGetPosition - ストリングの開始位置を戻す』
- 179 ページの『SQLGetStmtAttr - ステートメント属性の値の取得』
- 182 ページの『SQLGetStmtOption - ステートメント・オプションの現行設定を戻す』
- 184 ページの『SQLGetSubString - ストリング値の一部の検索』
- 187 ページの『SQLGetTypeInfo - データ・タイプ情報の入手』
- 192 ページの『SQLLanguages - SQL 言語または準拠情報の取得』
- 194 ページの『SQLMoreResults - さらに結果セットがあるかどうかの判別』
- 196 ページの『SQLNativeSql - 固有の SQL テキストの入手』
- 199 ページの『SQLNextResult - 次の結果セットの処理』
- 201 ページの『SQLNumParams - SQL ステートメント内のパラメーター数の入手』
- 203 ページの『SQLNumResultCols - 結果列の数の取得』
- 205 ページの『SQLParamData - データ値が必要な次のパラメーターの取得』
- 207 ページの『SQLParamOptions - パラメーターの入力配列の指定』
- 209 ページの『SQLPrepare - ステートメントの準備作成』
- 213 ページの『SQLPrimaryKeys - 表の基本キー列の入手』
- 216 ページの『SQLProcedureColumns - プロシージャーの入出力パラメーター情報の入手』
- 222 ページの『SQLProcedures - プロシージャー名リストの入手』
- 226 ページの『SQLPutData - パラメーターのデータ値に引き渡し』
- 229 ページの『SQLReleaseEnv - すべての環境リソースの解放』
- 231 ページの『SQLRowCount - 行数の取得』
- 233 ページの『SQLSetConnectAttr - 接続属性の設定』
- 238 ページの『SQLSetConnectOption - 接続オプションの設定』
- 240 ページの『SQLSetCursorName - カーソル名の設定』
- 242 ページの『SQLSetDescField - 記述子フィールドの設定』
- 244 ページの『SQLSetDescRec - 記述子レコードの設定』
- 246 ページの『SQLSetEnvAttr - 環境属性の設定』
- 251 ページの『SQLSetParam - パラメーターの設定』
- 252 ページの『SQLSetStmtAttr - ステートメント属性の設定』

- 256 ページの『SQLSetStmtOption - ステートメント・オプションの設定』
- 258 ページの『SQLSpecialColumns - 特殊な列 (行 ID) の取得』
- 262 ページの『SQLStatistics - 基本表の索引情報と統計情報の取得』
- 265 ページの『SQLTablePrivileges - 表に関連した特権の入手』
- 268 ページの『SQLTables - 表情報の取得』
- 271 ページの『SQLTransact - トランザクション管理』

SQLAllocConnect - 接続の割り振り

目的

SQLAllocConnect() は、接続ハンドルと、入力環境ハンドルによって識別される環境内の関連したリソースを割り振ります。いつでも割り振れる接続数を照会するには、 fInfoType を SQL_ACTIVE_CONNECTIONS に設定して、 SQLGetInfo() を呼び出してください。

この関数の前に、SQLAllocEnv() を呼び出す必要があります。

構文

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

関数引き数

表 6. SQLAllocConnect の引き数

データ・タイプ	引き数	使用	説明
SQLHENV	*henv*	入力	環境ハンドル
SQLHDBC *	*phdbc*	出力	接続ハンドルへのポインター

使用法

出力接続ハンドルは、DB2 UDB CLI によって使用されて、一般状況情報、トランザクション状態、およびエラー情報を含め、接続に関連するすべての情報が参照されます。

接続ハンドル (*phdbc*) へのポインターが、 SQLAllocConnect() によって割り振られた有効な接続ハンドルを指している場合は、この呼び出しによって元の値が上書きされます。これはアプリケーション・プログラミング・エラーであり、 DB2 UDB CLI では検出されません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQL_ERROR が戻された場合、 *phdbc* 引き数は SQL_NULL_HDBC に設定されます。 SQLError() は、環境変数 (*henv*) を指定し、 *hdbc* および *hstmt* 引き数をそれぞれ SQL_NULL_HDBC および SQL_NULL_HSTMT に設定して、アプリケーション・プログラムから呼び出す必要があります。

診断

表 7. SQLAllocConnect SQLSTATE

CLI SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	*phdbc* が NULL ポインターでした。

例

以下の例は、接続および環境に関する診断情報を得る方法を示しています。 SQLError() の使用例については、 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』の typical.c の完全なリストを参照してください。

コード例については、 viii ページの『コードの特記事項情報』を参照してください。

```
/*********************************************************************
** 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);

}
```

参照

- 30 ページの『SQLAllocEnv - 環境ハンドルの割り振り』
- 75 ページの『SQLConnect - データ・ソースへの接続』
- 90 ページの『SQLDisconnect - データ・ソースからの切断』
- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』
- 136 ページの『SQLGetConnectAttr - 接続属性の値の取得』
- 238 ページの『SQLSetConnectOption - 接続オプションの設定』

SQLAllocEnv - 環境ハンドルの割り振り

目的

SQLAllocEnv() は、環境ハンドルと関連したリソースを割り振ります。

この関数は、 SQLAllocConnect() その他の DB2 UDB CLI 関数よりも先に、アプリケーション・プログラムで呼び出す必要があります。 *henv* 値は、入力として環境ハンドルが必須になっているすべての後続呼び出しに渡されます。

構文

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

関数引き数

表 8. SQLAllocEnv の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV *	*phenv*	出力	環境ハンドルへのポインター

使用法

アプリケーション・プログラムごとに、活動状態の環境は常に 1 つのみです。 SQLAllocEnv() の後続の呼び出しからは、既存の環境ハンドルが戻されます。

デフォルトでは、 SQLFreeEnv() の最初の呼び出しが正常に完了すると、このハンドルに関連付けられているリソースが解放されます。 SQLAllocEnv() の呼び出しが何回正常に完了していても、この処理法は変わりません。環境属性 SQL_ATTR_ENVHNDL_COUNTER を SQL_TRUE に設定した場合、ハンドルに関連付けられているリソースが解放されるためには、その前に、 SQLAllocEnv() の呼び出しが正常完了するたびに 1 回ずつ SQLFreeEnv() を呼び出す必要があります。

すべての DB2 UDB CLI リソースを活動中にしておくには、 SQLAllocEnv() を呼び出すプログラムは、スタックを終了したり捨てたりしてはなりません。 そうすると、アプリケーションは、割り振ったオープン・カーソル、ステートメント・ハンドル、および他のリソースを失ってしまいます。

戻りコード

- SQL_SUCCESS
- SQL_ERROR

SQL_ERROR が戻され、*phenv* が SQL_NULL_HENV と等価である場合は、追加の診断情報を関連付けるためのハンドルがないため、 SQLError() は呼び出せません。

戻りコードが SQL_ERROR で環境ハンドルへのポインターが SQL_NULL_HENV と等価でない場合、このハンドルは**制限付きハンドル**になります。つまり、このハンドルを使用できるのは、より詳細なエラー情報を得るために SQLError() を呼び出す場合、または SQLFreeEnv() を呼び出す場合のみということになります。

診断

表 9. SQLAllocEnv SQLSTATE

SQLSTATE	説明	解説
58004	システム・エラー	リカバリー不能なシステム・エラーです。

例

コード例については、 viii ページの『コードの特記事項情報』を参照してください。

```
/*************************************************
** 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);
}
```

SQLAllocEnv

```
/*********************************************************************
**   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);
    }
}
```

参照

- 27 ページの『SQLAllocConnect - 接続の割り振り』
- 123 ページの『SQLFreeEnv - 環境ハンドルの解放』
- 35 ページの『SQLAllocStmt - ステートメント・ハンドルの割り振り』

SQLAllocHandle - ハンドルの割り振り

目的

SQLAllocHandle() は、あらゆるタイプのハンドルを割り振ります。

構文

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

関数引き数

表 10. SQLAllocHandle の引き数

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*htype*	入力	割り振るハンドルのタイプ。SQL_HANDLE_ENV、SQL_HANDLE_DBC、SQL_HANDLE_DESC、または SQL_HANDLE_STMT のいずれか。
SQLINTEGER	*ihandle*	入力	新しいハンドルを割り振るのに使うコンテキストを記述したハンドル。ただし、*htype* が SQL_HANDLE_ENV の場合、これは SQL_NULL_HANDLE になります。
SQLINTEGER *	ハンドル	出力	ハンドルへのポインター

使用法

この関数は、SQLAllocEnv()、 SQLAllocConnect()、および SQLAllocStmt() の組み合わせ関数になっています。

htype が SQL_HANDLE_ENV の場合、 *ihandle* は SQL_NULL_HANDLE でなければなりません。 *htype* が SQL_HANDLE_DBC の場合、 *ihandle* は有効な環境ハンドルでなければなりません。 *htype* が SQL_HANDLE_DESC または SQL_HANDLE_STMT の場合、 *ihandle* は有効な接続ハンドルでなければなりません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

引き数ハンドルが NULL ポインターであった場合、 SQL_ERROR が戻されます。

表 11. SQLAllocHandle SQLSTATE

SQLSTATE	説明	解説
58004	システム・エラー	リカバリー不能なシステム・エラーです。

SQLAllocHandle

表 11. SQLAllocHandle SQLSTATE (続き)

SQLSTATE	説明	解説
HY014	ハンドルが過多	最大数のハンドルがすでに割り振られています。

参照

- 27 ページの『SQLAllocConnect - 接続の割り振り』
- 30 ページの『SQLAllocEnv - 環境ハンドルの割り振り』
- 35 ページの『SQLAllocStmt - ステートメント・ハンドルの割り振り』

SQLAllocStmt - ステートメント・ハンドルの割り振り

目的

SQLAllocStmt() は、新規のステートメント・ハンドルを割り振り、このハンドルを、接続ハンドルで指定された接続に関連付けます。いつでも割り振りできるステートメント・ハンドル数に関して定義された制限事項はありません。

この関数の前に、SQLConnect() を呼び出す必要があります。

この関数は、SQLBindParam()、 SQLPrepare()、SQLExecute()、SQLExecDirect() その他、入力引き数の 1 つとしてステートメント・ハンドルを持つ関数より先に呼び出す必要があります。

構文

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

関数引き数

表 12. SQLAllocStmt の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル
SQLHSTMT *	*phstmt*	出力	ステートメント・ハンドルへのポインター

使用法

DB2 UDB CLI は、それぞれのステートメント・ハンドルを使用して、すべての記述子、結果値、カーソル情報、および状況情報を、処理される SQL ステートメントに関連させます。それぞれの SQL ステートメントにはステートメント・ハンドルがなければなりませんが、そのハンドルをさまざまなステートメントに再利用できます。

この関数を呼び出す場合は、 *hdbc* が活動中のデータベース接続を参照している必要があります。

位置の決まった更新または削除を実行するには、アプリケーション・プログラムが SELECT ステートメントおよび UPDATE または DELETE ステートメントごとに別々のステートメント・ハンドルを使用する必要があります。

ステートメント・ハンドル (*phstmt*) への入力ポインターが、 SQLAllocStmt() の前回の呼び出しで割り振られた有効なステートメント・ハンドルを指している場合は、この呼び出しの結果として元の値が上書きされます。これはアプリケーション・プログラミング・エラーであり、 DB2 UDB CLI では検出されません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLAllocStmt

SQL_ERROR が戻された場合、 *phstmt* 引き数は SQL_NULL_HSTMT に設定されます。 SQLError() は、同じ *hdbc* を指定し、 *hstmt* 引き数を SQL_NULL_HSTMT に設定してアプリケーション・プログラムから呼び出す必要があります。

診断

表 13. SQLAllocStmt SQLSTATE

SQLSTATE	説明	解説
08003	接続がオープンしていない	*hdbc* 引き数によって指定された接続はオープンしていませんでした。ドライバーで *hstmt* を割り振るには、接続が正常に確立されている (とともに接続がオープンしている) 必要があります。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	*phstmt* は NULL ポインターです。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

例

109 ページの『例』 SQLFetch() を参照してください。

参照

- 75 ページの『SQLConnect - データ・ソースへの接続』
- 127 ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』
- 182 ページの『SQLGetStmtOption - ステートメント・オプションの現行設定を戻す』
- 256 ページの『SQLSetStmtOption - ステートメント・オプションの設定』

SQLBindCol - アプリケーション・プログラム変数に対する列のバインド

目的

SQLBindCol() は、すべてのデータ・タイプを対象に、結果セットの列をアプリケーション・プログラム変数 (保管バッファー) に関連付け (バインド) ます。データは、 SQLFetch() の呼び出し時に DBMS からアプリケーション・プログラムに転送されます。

また、この関数は、必要な任意のデータ変換を指定する場合にも使用されます。この関数は、アプリケーション・プログラムで検索しなければならない結果セットの列ごとに 1 回ずつ呼び出します。

通常は、この関数より先に SQLPrepare() または SQLExecDirect() を呼び出します。また、SQLDescribeCol() または SQLColAttributes() を呼び出さなければならない場合もあります。

この呼び出しで指定した保管バッファーにデータを転送する場合は、 SQLFetch() よりも先に SQLBindCol() を呼び出してください。

構文

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

関数引き数

表 14. SQLBindCol の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*icol*	入力	列を識別する番号。列は、1 から始めて左から右へ順に番号が付けられています。

SQLBindCol

表 14. SQLBindCol の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*fCType*	入力	結果セットの列の番号 *icol* のアプリケーション・データ・タイプ。以下のタイプがサポートされています。 • 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 SQL_DEFAULT を指定すると、データがそのデフォルト・データ・タイプに転送されます。詳細については、 18 ページの表 3 を参照してください。
SQLPOINTER	*rgbValue*	出力 (据え置き)	取り出しの発生時に DB2 UDB CLI によって列データが保管されるバッファーへのポインター。 *rgbValue* が NULL の場合、列はアンバインドになっています。

表 14. *SQLBindCol* の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*cbValueMax*	入力	列データの保管に使用可能な *rgbValue* バッファーのサイズ (バイト単位)。 *fCType* が SQL_CHAR または SQL_DEFAULT の場合、 *cbValueMax* は > 0 になっている必要があり、それ以外ではエラーが戻されます。 *fcType* が SQL_DECIMAL または SQL_NUMERIC である場合、 *cbValueMax* は実際は精度と位取りでなければなりません。この 2 つの値を指定するには、 *(精度 * 256) + 位取り* を使います。またこれは、 SQLColAttributes() の使用時にこれらのデータ・タイプの長さとして戻される値でもあります。 *fcType* で任意の形式の 2 バイト文字を指定した場合は、 *cbValueMax* はバイト数ではなく 2 バイト文字の数でなければなりません。
SQLINTEGER *	*pcbValue*	出力 (据え置き)	*rgbValue* バッファーに戻す際に DB2 UDB CLI が使用可能なバイト数を示す値へのポインター。 この列のデータ値が NULL になっている場合、 SQLFetch() はこの引き数に SQL_NULL_DATA を戻します。この列のデータ値が、NULL 文字で終了するストリングで戻された場合、 SQL_NTS がこの引き数に戻されます。

注:

この関数の場合、 *rgbValue* と *pcbValue* の両方が据え置き出力になります。つまり、これらのポインターが指す保管場所は、 SQLFetch() が呼び出されるまで更新されないということです。これらのポインターが参照する場所は、 SQLFetch() が呼び出されるまでは有効になっている必要があります。

使用法

アプリケーション・プログラムは、検索したい結果セットの列ごとに SQLBindCol() を 1 回ずつ呼び出します。 SQLFetch() が呼び出されると、これらの各バインド 列のデータは、割り当てられている場所 (*rgbValue* および *pcbValue* ポインターにより指定されている) に保管されます。

まず SQLDescribeCol() または SQLColAttributes() を呼び出せば、アプリケーション・プログラムから、この列の属性 (データ・タイプ、長さなど) を照会できます。さらにこの情報を使用して、保管場所の正しいデータ・タイプを指定したり、他のデータ・タイプへのデータ変換を指示したりすることができます。詳細については、17 ページの『DB2 UDB CLI の関数でのデータ・タイプとデータ変換』を参照してください。

この後の取り出しのときに、アプリケーション・プログラムは、これらの列のバインドを変更したり、SQLBindCol() を呼び出してアンバインド列をバインドすることができます。新規のバインドは取り出されたデータには適用されず、 SQLFetch() の次回の呼び出しの時に使用されます。単一の列をアンバインドするには、 *rgbValue* を NULL に設定して SQLBindCol() を呼び出します。すべての列をアンバインドするには、 *fOption* 入力を SQL_UNBIND に設定して、アプリケーション・プログラムから SQLFreeStmt() を呼び出す必要があります。

列は、1 から始めて左から右へ順次割り当てられた番号で識別されます。結果セットの列の番号は、*fdescType* 引き数セットを SQL_DESC_COUNT に設定して SQLNumResultCols() または SQLColAttributes() を呼び出せば判別できます。

アプリケーション・プログラムの選択によっては、列はすべてバインドされるわけではないことも、あるいはまったくバインドされないこともあります。 SQLFetch() 呼び出しの後は、アンバインド列のデータ (アンバインド列だけ) を SQLGetData() で検索できます。 SQLGetData() よりも SQLBindCol() の方が効率的なので、可能であれば常にこちらを使用するようにしてください。

検索されるデータ用に十分な大きさのストレージが、アプリケーション・プログラムで確実に割り振られるようにする必要があります。バッファーに可変長データを保管する場合は、バインド列の最大長として必須になっている大きさのストレージを割り振る必要があり、そうでない場合データは切り捨てられます。

ストリングの切り捨てが行われると、SQL_SUCCESS_WITH_INFO が戻され、 *pcbValue* は、アプリケーション・プログラムに戻すのに使用できる *rgbValue* の実際のサイズに設定されます。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 15. SQLBindCol SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY002	列番号が無効	引き数 *icol* に指定された値が 0 になっていました。 引き数 *icol* に指定された値が、データ・ソースでサポートされている列の最大数を超過しました。
HY003	プログラム・タイプが範囲外	*fCType* は正しいデータ・タイプではありません。
HY009	引き数値が無効	*rgbValue* が NULL ポインターです。 引き数 *cbValueMax* に指定された値が 1 より小さくなっており、 *fCType* は SQL_CHAR か SQL_DEFAULT のどちらかになっています。

表 15. SQLBindCol SQLSTATE (続き)

SQLSTATE	説明	解説
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。
HY014	ハンドルが過多	最大数のハンドルがすでに割り振られていますが、この関数を使うには、さらに記述子ハンドルが必要です。
HYC00	ドライバーでサポートされていない	引き数 *fCType* に指定されているデータ・タイプは、ドライバーで認識はされますが、サポートされていません (**HY**003 も参照)。

例

109 ページの『例』SQLFetch() を参照してください。

参照

- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 108 ページの『SQLFetch - 次のデータ行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLBindFileToCol - LOB 列に対する LOB ファイル参照のバインド

目的

SQLBindFileToCol() は、結果セット内の LOB 列を、ファイル参照またはファイル参照配列に関連付ける (バインド) のに使います。そうすると、ステートメント・ハンドル用の各行の取り出し時に、その列内のデータを直接ファイルに転送することができます。

LOB ファイル参照の引き数 (ファイル名、ファイル名の長さ、ファイル参照オプション) は、アプリケーションの環境 (クライアント側の) 内のファイルを参照します。アプリケーションは、各行の取り出しの前に、ファイル名、ファイル名の長さ、およびファイル・オプション (new / overwrite / append) が、それらの変数内に入っていることを確かめる必要があります。この値は、取り出し時にそのつど変更することができます。

構文

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

関数引き数

表 16. SQLBindFileToCol 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLSMALLINT	*ColumnNumber*	入力	列を識別する番号。列は、1 から始めて左から右へ順に番号が付けられています。
SQLCHAR *	*FileName*	入力 (据え置き)	*StatementHandle* を使った次の取り出しのときに、ファイル名またはファイル名配列を保管する場所を指すポインター。これは、ファイルの完全パス名または相対ファイル名になります。相対ファイル名を使った場合、その名前は、実行中のアプリケーションの現行パスに追加されます。このポインターを NULL にすることはできません。
SQLSMALLINT *	*FileNameLength*	入力 (据え置き)	*StatementHandle* を使った次の取り出しのときに、ファイル名の長さ (または長さの配列) を保管する場所を指すポインター。このポインターが NULL の場合、 SQL_NTS という長さがとられます。 ファイル名の長さの最大値は 255 です。

表 16. SQLBindFileToCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	*FileOptions*	入力 (据え置き)	*StatementHandle* を使った次の取り出しのときに、ファイルに書き込むのに使うファイル・オプションを入れる場所を指すポインター。次のような *FileOptions* がサポートされています。 **SQL_FILE_CREATE** 新しいファイルを作成します。この名前の付いたファイルがすでに存在すると、SQL_ERROR が戻されます。 **SQL_FILE_OVERWRITE** このファイルがすでに存在すると、それを上書きします。存在しなければ、新しいファイルを作成します。 **SQL_FILE_APPEND** このファイルがすでに存在すると、それにデータを追加します。存在しなければ、新しいファイルを作成します。 ファイルごとに 1 つのオプションしか選択できません。デフォルト値はありません。
SQLSMALLINT	*MaxFileNameLength*	入力	*FileName* バッファーの長さを指定します。
SQLINTEGER *	*StringLength*	出力 (据え置き)	戻された LOB データのバイト単位の長さを入れる場所を指すポインター。このポインターが NULL の場合、何も戻されません。
SQLINTEGER *	*IndicatorValue*	出力 (据え置き)	標識値を入れる場所を指すポインター。

使用法

行の取り出しのときに、ファイルに直接転送する必要のある各列ごとに、アプリケーション・プログラムは SQLBindFileToCol() を 1 回ずつ呼び出します。 LOB データは、変換されたり NULL 終了文字を付加されたりしないで、ファイルに直接書き込まれます。

どの取り出しの前にも、*FileName*、*FileNameLength*、および *FileOptions* を設定していなければなりません。 SQLFetch() または SQLFetchScroll() を呼び出すと、 LOB ファイル参照にバインドされているすべての列のデータが、そのファイル参照が指し示す 1 つ以上のファイルに書き込まれます。
SQLBindFileToCol() の据え置き入力引き数値に関連したエラーが、取り出しのときに報告されます。 LOB ファイル参照と、据え置きの *StringLength* と *IndicatorValue* 出力引き数は、それぞれの取り出し操作の間に更新されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 17. SQLBindFileToCol SQLSTATE

SQLSTATE	説明	解説
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY002	列番号が無効	引き数 *icol* に指定された値が、1 未満になっていました。 引き数 *icol* に指定された値が、データ・ソースでサポートされている列の最大数を超過しました。
HY009	引き数値が無効	*FileName*、*StringLength*、または *FileOptions* が NULL ポインターです。
HY010	関数シーケンス・エラー	data-at-execute (`SQLParamData()`、 `SQLPutData()`) の操作中に関数を呼び出しました。 BEGIN COMPOUND および END COMPOUND SQL の操作中に関数を呼び出しました。
HY090	ストリングまたはバッファー長が無効	引き数 *MaxFileNameLength* に指定された値が 0 未満です。
HYC00	ドライバーでサポートされていない	現在、アプリケーション・プログラムは、ラージ・オブジェクトをサポートしないデータ・ソースに接続されています。

制約事項

ラージ・オブジェクト・データ・タイプをサポートしない DB2 サーバーに接続しているとき、この関数は使うことができません。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 108 ページの『SQLFetch - 次のデータ行』
- 45 ページの『SQLBindFileToParam - LOB パラメーターに対する LOB ファイル参照のバインド』

SQLBindFileToParam - LOB パラメーターに対する LOB ファイル参照のバインド

目的

SQLBindFileToParam() を使って、 SQL ステートメント内のパラメーター・マーカーを、ファイル参照またはファイル参照配列に関連付け (バインド) ます。それによって、該当するステートメントのその後の実行時に、ファイル内のデータを LOB 列に直接転送できるようになります。

LOB ファイル参照の引き数 (ファイル名、ファイル名の長さ、ファイル参照オプション) は、アプリケーションの環境 (クライアント側の) 内のファイルを参照します。アプリケーション・プログラムは、SQLExecute() または SQLExecDirect() を呼び出す前に、据え置き入力バッファー内のその情報を使用できることを確かめる必要があります。この値は、SQLExecute() の呼び出し時にそのつど変更することができます。

構文

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

関数引き数

表 18. SQLBindFileToParam 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLSMALLINT	*ParameterNumber*	入力	パラメーター・マーカーの番号。パラメーターは、1 から始めて左から右へ順に番号が付けられています。
SQLSMALLINT	*DataType*	入力	列の SQL データ・タイプ。データ・タイプは次のいずれかでなければなりません。 • SQL_BLOB • SQL_CLOB • SQL_DBCLOB
SQLCHAR *	*FileName*	入力 (据え置き)	ステートメント (*StatementHandle*) の実行時に、ファイル名またはファイル名配列を保管する場所を指すポインター。これは、ファイルの完全パス名または相対ファイル名になります。相対ファイル名を指定すると、クライアント・プロセスの現行パスにその名前が付加されます。 この引き数を NULL にすることはできません。

表 18. SQLBindFileToParam 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT *	*FileNameLength*	入力 (据え置き)	*StatementHandle* を使った次の SQLExecute() または SQLExecDirect() のときに、ファイル名の長さ (または長さの配列) を保管する場所を指すポインター。 このポインターが NULL の場合、 SQL_NTS という長さがとられます。 ファイル名の長さの最大値は 255 です。
SQLINTEGER *	*FileOptions*	入力 (据え置き)	ファイルの読み取り時に、ファイル・オプションまたはファイル・オプション配列を保管する場所を指すポインター。ステートメント (*StatementHandle*) の実行時に、この場所へのアクセスが行われます。 1 つのオプションだけがサポートされます (しかも指定する必要があります)。 **SQL_FILE_READ** オープン、読み取り、およびクローズすることのできる通常のファイル。 (その長さは、ファイルのオープン時に計算されます。) このポインターを NULL にすることはできません。
SQLSMALLINT	*MaxFileNameLength*	入力	*FileName* バッファーの長さを指定します。 アプリケーション・プログラムが、 SQLParamOptions() を呼び出してそれぞれのパラメーターに複数の値を指定した場合、これが *FileName* 配列内の各要素の長さになります。
SQLINTEGER *	*IndicatorValue*	出力 (据え置き)	標識値 (または値の配列) を入れる場所を指すポインター。パラメーターのデータ値が NULL になることになっている場合は、これは SQL_NULL_DATA に設定されます。データ値が NULL でない場合は、これを 0 に設定しなければなりません (または、ポインターを NULL に設定することもできます)。

使用法

ステートメントの実行時にファイルから直接取得する必要のある値をもつパラメーター・マーカーごとに、アプリケーション・プログラムは SQLBindFileToParam() を 1 回ずつ呼び出します。そのステートメントの実行の前に、 *FileName*、*FileNameLength*、および *FileOptions* 値を設定しておかなければなりません。そのステートメントの実行時、 SQLBindFIleToParam() を使ってバインドされたすべてのパラメーターが、参照ファイルから読み取られて、サーバーに渡されます。

LOB パラメーター・マーカーを、 SQLBindFileToParam() を使って入力ファイルに関連付け (バインド) たり、 SQLBindParameter() を使って保管バッファーに関連付けたりすることができます。バインド・パラメーター関数の最新の呼び出しで、有効になっているバインドのタイプが判別されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 19. SQLBindFileToParam SQLSTATE

SQLSTATE	説明	解説
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY004	SQL データ・タイプが範囲外	*DataType* に指定された値は、この関数呼び出しでは有効な SQL タイプではありません。
HY009	引き数値が無効	*FileName*、*FileOptions*、 *FileNameLength*、が NULL ポインターです。
HY010	関数シーケンス・エラー	data-at-execute (`SQLParamData()`、 `SQLPutData()`) の操作中に関数を呼び出しました。 BEGIN COMPOUND および END COMPOUND SQL の操作中に関数を呼び出しました。
HY090	ストリングまたはバッファー長が無効	入力引き数 *MaxFileNameLength* に指定された値は 0 未満です。
HY093	パラメーターの数値が無効	*ParameterNumber* に指定した値が、1 未満であるか、またはサポートされている最大パラメーター数より多いです。
HYC00	ドライバーでサポートされていない	サーバーは、ラージ・オブジェクト・データ・タイプをサポートしません。

制約事項

ラージ・オブジェクト・データ・タイプをサポートしない DB2 サーバーに接続しているとき、この関数は使うことができません。

参照

- 48 ページの『SQLBindParam - パラメーター・マーカーに対するバッファーのバインド』
- 103 ページの『SQLExecute - ステートメントの実行』
- 207 ページの『SQLParamOptions - パラメーターの入力配列の指定』

SQLBindParam - パラメーター・マーカーに対するバッファーのバインド

目的

SQLBindParam() は、アプリケーション・プログラム変数を SQL ステートメントのパラメーター・マーカーにバインドします。また、この関数を使って、アプリケーション・プログラム変数を、パラメーターが入出力されるストアード・プロシージャー CALL ステートメントのパラメーターにバインドすることもできます。この関数は SQLSetParam() と同じです。

構文

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

関数引き数

表 20. SQLBindParam の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*ipar*	入力	パラメーター・マーカー番号。1 から始めて左から右へ順に番号付けされています。

表 20. *SQLBindParam* の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*fCType*	入力	パラメーターのアプリケーション・データ・タイプ。以下のタイプがサポートされています。 • 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 SQL_DEFAULT を指定すると、データがそのデフォルトのアプリケーション・データ・タイプから、 *fSqlType* に示されているタイプに転送されます。

SQLBindParam

表 20. SQLBindParam の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*fSqlType*	入力	パラメーターの SQL データ・タイプ。サポートされているタイプは以下のとおりです。 • 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*	入力	対応するパラメーター・マーカーの精度。*fSqlType* が以下の値になっている場合、意味は次のとおりです。 • 単一バイト文字列 (SQL_CHAR など)。このパラメーターに送信されるバイト単位の最大長を表します。この長さには、NULL 終了文字も含まれます。 • 2 バイト文字列 (SQL_GRAPHIC など)。このパラメーターの 2 バイト文字の最大長を表します。 • SQL_DECIMAL または SQL_NUMERIC。小数部の最大精度を表します。 • この他の場合、この引き数は使用されません。

表 20. SQLBindParam の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*ibScale*	入力	*fSqlType* が SQL_DECIMAL または SQL_NUMERIC である場合は、対応するパラメーターの位取り。 *fSqlType* が SQL_TIMESTAMP である場合は、この値がタイム・スタンプの文字表示の 10 進小数点の右側の桁数になります (たとえば、yyyy-mm-dd hh:mm:ss.fff の位取りは 3)。 この部分で説明した *fSqlType* 値の場合以外、*ibScale* は使用されません。
SQLPOINTER	*rgbValue*	入力 (据え置き) または 出力 (据え置き)	実行の時点で、 *pcbValue* に SQL_NULL_DATA も SQL_DATA_AT_EXEC も入っていない場合、 *rgbValue* はパラメーターの実際のデータが入っているバッファーを指します。 *pcbValue* に SQL_DATA_AT_EXEC が入っている場合、 *rgbValue* はこのパラメーターに関連するアプリケーション・プログラム定義の 32 ビット値になります。この 32 ビット値は、あとで `SQLParamData()` 呼び出しのときにアプリケーション・プログラムに戻されます。
SQLINTEGER *	*pcbValue*	入力 (据え置き) または出力 (据え置き) あるいはその両方	ステートメントの実行時に値を変換される変数。 • NULL 値がパラメーターとして使用される場合、 *pcbValue* の値は SQL_NULL_DATA になっている必要があります。 • `ParamData()` および `PutData()` 呼び出しにより実行時に動的引き数が指定される場合、 *pcbValue* の値は SQL_DATA_AT_EXEC になっている必要があります。 • *fcType* が SQL_CHAR で、 *rgbValue* のデータに NULL 文字終了ストリングがある場合、 *pcbValue* の値は *rgbValue* の長さか SQL_NTS 値になっている必要があります。 • *fcType* が SQL_CHAR で、 *rgbValue* のデータが NULL 文字で終了していない場合、 *pcbValue* の値は *rgbValue* のデータの長さになっている必要があります。 • *fcType* が LOB タイプの場合、 *pcbValue* の値は *rgbValue* のデータの長さになっている必要があります。 • その他の場合、*pcbValue* はゼロでなければなりません。

使用法

SQLBindParam() をアプリケーション・プログラム変数のストアード・プロシージャーの出力パラメーターへのバインドに使用する場合に、 *rgbValue* バッファーがメモリーの *pcbValue* バッファーに後続して保管されていると、 DB2 UDB CLI ではパフォーマンスが多少向上します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 21. SQLBindParam SQLSTATE

SQLSTATE	説明	解説
07006	制限付きデータ・タイプ属性違反	SQLSetParam() と同じ。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY003	プログラム・タイプが範囲外	SQLSetParam() と同じ。
HY004	SQL データ・タイプが範囲外	SQLSetParam() と同じ。
HY009	引き数値が無効	*rgbValue*、*pcbValue* が NULL、または *ipar* が 1 未満でした。
HY010	関数シーケンス・エラー	SQLExecute() または SQLExecDirect() から SQL_NEED_DATA が戻された後に関数が呼び出されましたが、すべての *data-at-execution* パラメーター用のデータが送信されたわけではありません。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。
HY014	ハンドルが過多	最大数のハンドルがすでに割り振られています。

SQLBindParameter - バッファーに対するパラメーター・マーカーのバインド

目的

SQLBindParameter() は、SQL ステートメント内のパラメーター・マーカーを、アプリケーション・プログラム変数に関連付ける (バインドする) のに使います。データは、 SQLExecute() または SQLExecDirect() の呼び出し時にアプリケーション・プログラムから DBMS に転送されます。データが転送されるときに、データ変換が行われることがあります。

また、この関数を使って、アプリケーション・プログラム・ストレージを、パラメーターの入力と出力の片方または両方が行われるストアード・プロシージャー CALL ステートメントのパラメーターにバインドしなければなりません。基本的にこの関数は、SQLSetParam() の拡張です。

構文

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

関数引き数

表 22. SQLBindParameter 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル
SQLSMALLINT	ParameterNumber	入力	パラメーター・マーカー番号。1 から始めて左から右へ順に番号付けされています。

SQLBindParameter

表 22. SQLBindParameter 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	InputOutputType	入力	パラメーターのタイプです。 IPD の SQL_DESC_PARAMETER_TYPE フィールドの値も、この引き数に設定されます。サポートされているタイプは以下のとおりです。 • SQL_PARAM_INPUT - パラメーター・マーカーは、ストアード・プロシージャー呼び出しではない SQL ステートメントに関連付けられます。あるいは、呼び出されるストアード・プロシージャーのパラメーターにマークを付けます。 ステートメントの実行時に、このパラメーターの実際のデータ値がサーバーに送られます。 *ParameterValuePtr* バッファーには、有効な入力データ値が入っていなければなりません。 *StrLen_or_IndPtr* バッファーには、それに対応する長さの値か、または SQL_NTS、SQL_NULL_DATA、または (SQLParamData() および SQLPutData() を介して値を送る必要がある場合は) SQL_DATA_AT_EXEC が入っていなければなりません。 • SQL_PARAM_INPUT_OUTPUT - パラメーター・マーカーは、呼び出されるストアード・プロシージャーの入出力パラメーターに関連付けられます。 ステートメントの実行時に、このパラメーターの実際のデータ値がサーバーに送られます。 *ParameterValuePtr* バッファーには、有効な入力データ値が入っていなければなりません。 *StrLen_or_IndPtr* バッファーには、それに対応する長さの値か、または SQL_NTS、SQL_NULL_DATA、または (SQLParamData() および SQLPutData() を介して値を送る必要がある場合は) SQL_DATA_AT_EXEC が入っていなければなりません。 • SQL_PARAM_OUTPUT - パラメーター・マーカーは、呼び出されるストアード・プロシージャーの出力パラメーターに関連付けられるか、またはストアード・プロシージャーの戻り値に関連付けられます。 ステートメントの実行後、出力パラメーター用のデータは、 *ParameterValuePtr* および *StrLen_or_IndPtr* で指定されたアプリケーション・バッファーに戻されます。ただし、どちらも NULL ポインターでない場合に限ります。どちらも NULL ポインターの場合は、出力データは廃棄されます。出力が戻り値をもっていない場合、 *StrLen_or_IndPtr* は SQL_NULL_DATA に設定されます。

表 22. *SQLBindParameter* 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	ValueType	入力	パラメーターの C データ・タイプ。以下のタイプがサポートされています。 • 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 SQL_C _DEFAULT を指定すると、データがそのデフォルトの C データ・タイプから、 *ParameterType* に指示されているタイプに転送されます。
SQLSMALLINT	ParameterType	入力	パラメーターの SQL データ・タイプ。
SQLINTEGER	ColumnSize	入力	対応するパラメーター・マーカーの精度。 *ParameterType* が以下の値になっている場合、意味は次のとおりです。 • 2 バイトまたは単一バイト文字列 (たとえば SQL_CHAR)。このパラメーター・マーカーのバイト単位の最大長を表します。 • 2 バイト文字列 (たとえば SQL_GRAPHIC)。このパラメーターの 2 バイト文字の最大長を表します。 • SQL_DECIMAL または SQL_NUMERIC。小数部の最大精度を表します。 • その他の場合、この引き数は無視されます。

SQLBindParameter

表 22. SQLBindParameter 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	DecimalDigits	入力	*ParameterType* が SQL_DECIMAL または SQL_NUMERIC の場合は、対応するパラメーターの位取りです。 *ParameterType* が SQL_TYPE_TIMESTAMP の場合は、この値が、タイム・スタンプの文字表示における小数点の右側の桁数になります (たとえば、 yyyy-mm-dd hh:mm:ss.fff の位取りは 3)。 上記の *ParameterType* 値の場合以外、 *DecimalDigits* は無視されます。
SQLPOINTER	ParameterValuePtr	入力 (据え置き) および出力 (据え置き) の片方または両方	• 入力 (*InputOutputType* を SQL_PARAM_INPUT または SQL_PARAM_INPUT_OUTPUT に設定) では、次のようになります。 実行時に、 *StrLen_or_IndPtr* に SQL_NULL_DATA も SQL_DATA_AT_EXEC も入っていない場合、 *ParameterValuePtr* は、パラメーターの実際のデータが入っているバッファーを指します。 *StrLen_or_IndPtr* に SQL_DATA_AT_EXEC が入っている場合、 *ParameterValuePtr* は、このパラメーターに関連したアプリケーション・プログラム定義の 32 ビット値になります。この 32 ビット値は、その後の `SQLParamData()` 呼び出しのときにアプリケーション・プログラムに戻されます。 パラメーターに複数の値を指定するために `SQLParamOptions()` が呼び出された場合、 *ParameterValuePtr* は、 *BufferLength* バイトの入力バッファー配列を指すポインターになります。 • 出力 (*InputOutputType* を SQL_PARAM_OUTPUT または SQL_PARAM_INPUT_OUTPUT に設定) では、次のようになります。 *ParameterValuePtr* は、ストアード・プロシージャーの出力パラメーター値を保管するバッファーを指します。 *InputOutputType* を SQL_PARAM_OUTPUT に設定した場合に、 *ParameterValuePtr* と *StrLen_or_IndPtr* がどちらも NULL ポインターであると、出力パラメーター値またはストアード・プロシージャーからの戻り値は廃棄されます。
SQLINTEGER	BufferLength	入力	使用されません。

表 22. *SQLBindParameter* 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	StrLen_or_IndPtr	入力 (据え置き) および出力 (据え置き) の片方または両方	これが入力または入出力パラメーターの場合、次のようになります。 これは、 *ParameterValuePtr* で保管されるパラメーター・マーカー値の長さの入った場所を指すポインター (ステートメントの実行時) になります。 パラメーター・マーカーに NULL 値を指定するには、この保管場所に SQL_NULL_DATA が入っていなければなりません。 *ValueType* が SQL_C_CHAR である場合、この保管場所には、 *ParameterValuePtr* で保管されている正確な長さが入っていなければなりません。ただし、 *ParameterValuePtr* が NULL 文字で終了している場合は、 SQL_NTS が入っていなければなりません。 *ValueType* が LOB データを示している場合、この保管場所には *ParameterValuePtr* で保管されるデータの長さが入っていなければなりません。 *ValueType* が文字データを指示している (明示的に、または SQL_C_DEFAULT を使って暗黙で) 場合に、このポインターを NULL に設定すると、アプリケーション・プログラムは常に *ParameterValuePtr* に、ヌル終了ストリングを提供するものと見なされます。これは、このパラメーター・マーカーは決して NULL 値をもたないことも意味します。 SQLExecute() または SQLExecDirect() を呼び出したときに、 *StrLen_or_IndPtr* が SQL_DATA_AT_EXEC の値を指していると、そのパラメーターのデータは SQLPutData() で送信されます。このパラメーターを、**data-at-execution** パラメーターと呼びます。

使用法

パラメーター・マーカーは、 SQL ステートメントでは "?" 文字で表され、このステートメントの実行時に、アプリケーション・プログラムから指定された値に置き換える桁のステートメント内の位置を指示するのに使われます。この値は、アプリケーション・プログラム変数から取り込みます。

アプリケーション・プログラムは、SQL ステートメントの実行の前に、その SQL ステートメント内の各パラメーター・マーカーに変数をバインドしなければなりません。この関数では、 *ParameterValuePtr* と *StrLen_or_IndPtr* が据え置き引き数です。ステートメントの実行時、保管場所は、有効になっていて入力データ値が入っていなければなりません。つまり、 SQLExecDirect() または SQLExecute() 呼び出しを、SQLBindParameter() 呼び出しと同じプロシージャー有効範囲内にとどめておく、あるいは、これらの保管場所を動的に割り振るか、静的またはグローバルに宣言する必要があるということです。

SQLBindParameter

パラメーター・マーカーは、 1 から始めて左から右へ順に付けられた番号 (*ParameterNumber*) で参照されます。

SQL_DROP または SQL_RESET_PARAMS オプションを指定して SQLFreeStmt() を呼び出すまで、または同じパラメーター *ParameterNumber* 番号で SQLBindParameter() をもう一度呼び出すまで、この関数でバインドされたすべてのパラメーターは有効のままになります。

SQL ステートメントの実行が完了し、その結果が処理された後、アプリケーション・プログラムは、別の SQL ステートメントを実行するのにこのステートメント・ハンドルを再利用するのがよいかもしれません。パラメーター・マーカーの指定が異なる (パラメーター数、長さ、タイプ) 場合、SQL_RESET_PARAMS を指定した SQLFreeStmt() を呼び出して、パラメーターのバインドをリセットまたは切断しなければなりません。

ValueType で指定する C バッファー・データ・タイプは、 *ParameterType* で指示する SQL データ・タイプと互換性がなければなりません。そうでない場合、エラーが起きます。

アプリケーション・プログラムは、パラメーター値を *ParameterValuePtr* バッファーに入れるか、または SQLPutData() を複数回呼び出して、この値を渡すことができます。 後者の場合、そのパラメーターは data-at-execution パラメーターになります。アプリケーションから DB2 UDB CLI に data-at-execution パラメーターを知らせるには、 *StrLen_or_IndPtr* バッファー内に SQL_DATA_AT_EXEC 値を入れます。そうすると、 *ParameterValuePtr* 入力引き数は 32 ビット値に設定され、それは、その後の SQLParamData() 呼び出しで戻されます。またこれを、パラメーター位置を識別するのに使用することができます。

ステートメントが実行されない限り、 *ParameterValuePtr* および *StrLen_or_IndPtr* によって参照される変数は検査されないので、 SQLExecute() または SQLExecDirect() を呼び出さない限り、データの内容や形式のエラーは検出も報告もされません。

基本的に SQLBindParameter() は、パラメーターが入力、入出力、または出力のどれかを指定する手段を提供することで、 SQLSetParam() 関数の機能を拡張します。この情報は、ストアード・プロシージャー用のパラメーターを正しく処理するために必要です。

InputOutputType 引き数は、パラメーターのタイプを指定します。 SQL ステートメント中の、プロシージャーを呼び出さないすべてのパラメーターは、入力パラメーターになります。ストアード・プロシージャー呼び出し内のパラメーターは、入力、入出力、または出力パラメーターのいずれかになります。通常、DB2 のストアード・プロシージャーの引き数の規則では、すべてのプロシージャー引き数は入出力であることが暗黙で了解されていますが、アプリケーション・プログラマーの選択によっては、 SQLBindParameter() 上で入力または出力の特性をさらに厳密に指定して、より積極的なコーディング・スタイルを実現することができます。ただし、そのタイプは、 SQL CREATE PROCEDURE ステートメントでストアード・プロシージャーを登録した際に指定したパラメーター・タイプと整合していなければなりません。

- アプリケーション・プログラムが、プロシージャー呼び出し内のパラメーターのタイプを判別できない場合、 *InputOutputType* を SQL_PARAM_INPUT に設定してください。データ・ソースがそのパラメーターの値を戻した場合、 DB2 UDB CLI はそれを破棄します。
- アプリケーション・プログラムで、パラメーターに SQL_PARAM_INPUT_OUTPUT または SQL_PARAM_OUTPUT のマークを付けた場合に、データ・ソースから値が戻されないと、 DB2 UDB CLI は *StrLen_or_IndPtr* バッファーを SQL_NULL_DATA に設定します。
- アプリケーション・プログラムがパラメーターに SQL_PARAM_OUTPUT のマークを付けた場合、CALL ステートメントの処理後にそのパラメーターのデータがアプリケーション・プログラムに戻されます。 *ParameterValuePtr* と *StrLen_or_IndPtr* 引き数がどちらも NULL ポインターの場合、 DB2 UDB

CLI は出力値を破棄します。 出力パラメーターの値がデータ・ソースから戻されないと、 DB2 UDB CLI は *StrLen_or_IndPtr* バッファーを SQL_NULL_DATA に設定します。

- この関数の場合、 *ParameterValuePtr* と *StrLen_or_IndPtr* はどちらも据え置き引き数です。 *InputOutputType* が SQL_PARAM_INPUT または SQL_PARAM_INPUT_OUTPUT に設定されている場合、ステートメントの実行時、保管場所は、有効になっていて入力データ値が入っていなければなりません。つまり、 SQLExecDirect() または SQLExecute() 呼び出しを、 SQLBindParameter() 呼び出しと同じプロシージャー有効範囲内にとどめておく、あるいは、これらの保管場所を動的に割り振るか、静的/グローバルに宣言する必要があるということです。

  同様に、 *InputOutputType* が SQL_PARAM_OUTPUT または SQL_PARAM_INPUT_OUTPUT に設定されている場合、 CALL ステートメントの実行が完了するまで、 ParameterValuePtr バッファーと StrLen_or_IndPtr バッファーの場所は有効のままでなければなりません。

アプリケーション・プログラムは、パラメーター値を *ParameterValuePtr* バッファーに入れるか、または SQLPutData() を複数回呼び出して、この値を渡すことができます。 後者の場合、そのパラメーターは data-at-execution パラメーターになります。アプリケーション・プログラムから DB2 UDB CLI に data-at-execution パラメーターを知らせるには、 StrLen_or_IndPtr バッファー内に SQL_DATA_AT_EXEC 値を入れます。 そうすると、 *ParameterValuePtr* 入力引き数は 32 ビット値に設定され、それは、その後の SQLParamData() 呼び出しで戻されます。またこれを、パラメーター位置を識別するのに使用することができます。

SQLBindParameter() を使って、アプリケーション・プログラム変数を、ストアード・プロシージャーの出力パラメーターにバインドする場合に、 *StrLen_or_IndPtr* バッファーの後のメモリー中で、 *ParameterValuePtr* バッファーが連続して置かれていると、 DB2 UDB CLI のパフォーマンスが多少向上することがあります。以下に例を示します。

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

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 23. SQLBindParameter SQLSTATE

SQLSTATE	説明	解説
07006	変換は無効	*ValueType* 引き数で指定されたデータ値から、 *ParameterType* 引き数で指定されたデータ・タイプへの変換は、有意義な変換ではありません。 (たとえば、SQL_C_DATE から SQL_DOUBLE への変換。)
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 23. SQLBindParameter SQLSTATE (続き)

SQLSTATE	説明	解説
HY003	プログラム・タイプが範囲外	引き数 *ParameterNumber* で指定された値は、有効なデータ・タイプでも SQL_C_DEFAULT でもありません。
HY004	SQL データ・タイプが範囲外	*ParameterType* に指定された値は有効な SQL データ・タイプではありません。
HY009	引き数値は無効	引き数 *ParameterValuePtr* は NULL ポインターで、引き数 *StrLen_or_IndPtr* も NULL ポインターですが、 *InputOutputType* は SQL_PARAM_OUTPUT ではありません。
HY010	関数シーケンス・エラー	SQLExecute() または SQLExecDirect() から SQL_NEED_DATA が戻された後に関数が呼び出されましたが、すべての *data-at-execution* パラメーターにデータが送信されていません。
HY013	予想外のメモリー処理エラー	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーにアクセスできません。
HY014	ハンドルが過多	最大数のハンドルがすでに割り振られています。
HY021	記述子情報の不整合	整合性検査で検査された記述子情報には一貫性がありませんでした。
HY090	ストリングまたはバッファー長が無効	引き数 *BufferLength* に指定された値が 0 未満です。
HY093	パラメーター番号は無効	引き数 *ValueType* に指定した値は、1 未満であるか、またはサーバーでサポートされている最大パラメーター数より多いです。
HY094	位取り値は無効	*ParameterType* に指定した値は SQL_DECIMAL または SQL_NUMERIC ですが、 *DecimalDigits* に指定した値は、0 未満であるか、または引き数 *ParamDef* (精度) より小さいです。 *ParameterType* に指定した値は SQL_C_TIMESTAMP で、*ParameterType* の値は SQL_CHAR または SQL_VARCHAR ですが、 *DecimalDigits* の値は、0 未満であるか、または 6 より大きいです。
HY104	精度値は無効	*ParameterType* に指定した値は SQL_DECIMAL または SQL_NUMERIC ですが、 *ParamDef* に指定した値は 1 未満です。
HY105	パラメーター・タイプは無効	*InputOutputType* は、 SQL_PARAM_INPUT、SQL_PARAM_OUTPUT、または SQL_PARAM_INPUT_OUTPUT のいずれでもありません。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI またはデータ・ソースは、引き数 *ValueType* に指定された値と、引き数 *ParameterType* に指定された値を組み合わせて指定された変換をサポートしていません。 引き数 *ParameterType* に指定された値は、 DB2 UDB CLI でもデータ・ソースでもサポートされていません。

参照

- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 205 ページの『SQLParamData - データ値が必要な次のパラメーターの取得』
- 226 ページの『SQLPutData - パラメーターのデータ値に引き渡し』

SQLCancel - ステートメントの取り消し

目的

SQLCancel() は、非同期で実行中の SQL ステートメントの操作処理の終了を試みます。

SQLCancel() は互換性のためにあるだけで、 SQL ステートメント実行に対して何の効果もありません。

構文

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

関数引き数

表 24. SQLCancel の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル

使用法

正常完了戻りコードは、取り消し要求が実装システムで受け入れられたことを示しますが、処理が取り消されるとは限りません。

戻りコード

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

診断

表 25. SQLCancel SQLSTATE

SQLSTATE	説明	解説
HY009 *	引き数値が無効	*hstmt* はステートメント・ハンドルではありません。

制約事項

DB2 UDB CLI では、非同期ステートメント実行はサポートされていません。

SQLCloseCursor - カーソル・ステートメントのクローズ

目的

SQLCloseCursor() は、ステートメント・ハンドル上のオープン・カーソルをクローズします。

構文

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

関数引き数

表 26. SQLCancel の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル

使用法

SQLCloseCursor() を呼び出すと、このステートメント・ハンドルに関連したカーソルがすべてクローズされ、保留中の結果も廃棄されます。このステートメント・ハンドルに関連するオープン・カーソルがない場合、この関数の効果はありません。

ステートメント・ハンドルが、複数の結果セットを含むストアード・プロシージャーを参照している場合、SQLCloseCursor() は現行の結果セットだけクローズします。それ以外の結果セットはすべてオープンしたままで、使用可能です。

戻りコード

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

診断

表 27. SQLCancel SQLSTATE

SQLSTATE	説明	解説
08003 *	接続がオープンしていない	hstmt の接続が確立されていません。
HY009 *	引き数値が無効	*hstmt* はステートメント・ハンドルではありません。

SQLColAttributes - 列属性

目的

SQLColAttributes() は、結果セットの列の属性を取得しますが、列の数を判別するのにも使用されます。SQLColAttributes() は、 SQLDescribeCol() 関数を拡張した代替関数です。

この関数の前に、 SQLPrepare() と SQLExecDirect() のどちらかを呼び出す必要があります。

この列のさまざまな属性 (データ・タイプ、長さなど) がアプリケーション・プログラムで認識されていない場合、 SQLBindCol() の前にこの関数 (または SQLDescribeCol()) を呼び出す必要があります。

構文

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

関数引き数

表 28. SQLColAttributes の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*icol*	入力	結果セット内の列番号 (1 から結果セットの列数までの範囲でなければなりません)。SQL_DESC_COUNT を指定すると、この引き数は無視されます。
SQLSMALLINT	*fDescType*	入力	サポートされている値は、表 29 に説明されています。
SQLCHAR *	*rgbDesc*	出力	ストリング列属性のバッファーへのポインター
SQLINTEGER	*cbDescMax*	入力	記述子バッファーの長さ (*rgbDesc*)
SQLINTEGER *	*pcbDesc*	出力	記述子の中の実際に戻すバイト数。この引き数の値が *rgbDesc* バッファーの長さに等しいかまたはそれより長い場合、値は切り捨てられています。この場合、記述子は *cbDescMax* から 1 バイト減算した数値に切り捨てられます。
SQLINTEGER *	*pfDesc*	出力	数値列属性に関する情報が入っている整数へのポインター。

表 29. fDescType 記述子タイプ

記述子	タイプ	説明
SQL_DESC_COUNT	SMALLINT	結果セットの列の数は、*pfDesc* に戻されます。

表 29. fDescType 記述子タイプ (続き)

記述子	タイプ	説明
SQL_DESC_NAME	CHAR(128)	列 *icol* の名前は、*rgbDesc* に戻されます。列が式である場合、戻される結果は製品固有になります。
SQL_DESC_TYPE	SMALLINT	*icol* で識別される列の SQL データ・タイプは、 *pfDesc* に戻されます。*pfSqlType* の有効値は、 19 ページの表 5 にリストされています。
SQL_DESC_LENGTH	INTEGER	*pfDesc* には、列に関連したデータのバイト 数が戻されます。 *icol* で識別される列が文字ベース、たとえば SQL_CHAR、SQL_VARCHAR、または SQL_LONG_VARCHAR である場合、実際のまたは最大の長さが戻されます。 列タイプが SQL_DECIMAL または SQL_NUMERIC であると、SQL_DESC_LENGTH は *(精度 * 256) + 位取り* になります。これは、同じ値が `SQLBindCol()` でも入力として渡せるように戻されます。精度と位取りは、これらのデータ・タイプごとに別々の値として取得することができます。それには、SQL_DESC_PRECISION と SQL_DESC_SCALE を使用します。
SQL_DESC_PRECISION	SMALLINT	列の精度属性が戻されます。
SQL_DESC_SCALE	SMALLINT	列の位取り属性が戻されます。
SQL_DESC_NULLABLE	SMALLINT	*icol* で識別される列で NULL が有効である場合、 *pfDesc* には SQL_NULLABLE が戻されます。 列制約で NULL が受け入れられない場合、*pfDesc* には SQL_NO_NULLS が戻されます。
SQL_DESC_UNNAMED	SMALLINT	これは、NAME フィールドが実際の名前である場合は SQL_NAMED ですが、 NAME フィールドが実装システム生成名である場合は SQL_UNNAMED です。
SQL_DESC_AUTO_INCREMENT	INTEGER	新しい行を表に挿入するたびに列を自動的に増分できる場合は SQL_TRUE です。列を自動的に増分できない場合は SQL_FALSE です。

表 29. fDescType 記述子タイプ (続き)

記述子	タイプ	説明
SQL_DESC_SEARCHABLE	INTEGER	**WHERE** 文節内で列を使用できない場合は SQL_UNSEARCHABLE です。 **LIKE** 述部を付けた場合にのみ **WHERE** 文節内で列を使用できる場合は、SQL_LIKE_ONLY です。 **WHERE** 文節内で、列を **LIKE** 以外のすべての比較演算子と一緒に使用できる場合は、SQL_ALL_EXCEPT_LIKE です。 **WHERE** 文節内で、列をどの比較演算子とでも一緒に使用できる場合は、SQL_SEARCHABLE です。 この属性を検索するには、ステートメント・ハンドルと接続ハンドルのどちらかについて、属性 SQL_ATTR_EXTENDED_COL_INFO が SQL_TRUE に設定されていなければなりません。
SQL_DESC_UPDATABLE	INTEGER	列は、定義された定数の値によって記述されます。 SQL_ATTR_READONLY SQL_ATTR_WRITE SQL_ATTR_READWRITE_UNKNOWN SQL_COLUMN_UPDATABLE は、結果セット内の列が更新可能かどうかを記述します。列が更新可能かどうかは、データ・タイプ、ユーザー特権、および結果セット自体の定義に基づいて決まる場合があります。列が更新可能かどうかが不明確な場合は、SQL_ATTR_READWRITE_UNKNOWN が戻されることになります。 この属性を検索するには、ステートメント・ハンドルと接続ハンドルのどちらかについて、属性 SQL_ATTR_EXTENDED_COL_INFO が SQL_TRUE に設定されていなければなりません。

表 29. fDescType 記述子タイプ (続き)

記述子	タイプ	説明
SQL_DESC_BASE_TABLE	CHAR(128)	この列が作成される基礎表の名前。 この属性を検索するには、ステートメント・ハンドルと接続ハンドルのどちらかについて、属性 SQL_ATTR_EXTENDED_COL_INFO が SQL_TRUE に設定されていなければなりません。
SQL_DESC_BASE_COLUMN	CHAR(128)	この列が作成される基礎表の中の実際の列の名前。 この属性を検索するには、ステートメント・ハンドルと接続ハンドルのどちらかについて、属性 SQL_ATTR_EXTENDED_COL_INFO が SQL_TRUE に設定されていなければなりません。
SQL_DESC_BASE_SCHEMA	CHAR(128)	この列が作成される基礎表のスキーマ名。 この属性を検索するには、ステートメント・ハンドルと接続ハンドルのどちらかについて、属性 SQL_ATTR_EXTENDED_COL_INFO が SQL_TRUE に設定されていなければなりません。
SQL_DESC_LABEL	CHAR(128)	この列のラベル (存在する場合)。存在しなければ、ゼロ長のストリング。 この属性を検索するには、ステートメント・ハンドルと接続ハンドルのどちらかについて、属性 SQL_ATTR_EXTENDED_COL_INFO が SQL_TRUE に設定されていなければなりません。

使用法

SQLDescribeCol() は特定の一連の引き数を戻しますが、 SQLColAttributes() を使うと、入手したい特定の列の特定の属性を指定することができます。入手したい情報がストリングである場合は、 *rgbDesc* に戻されます。入手したい情報が数値である場合は、 *pfDesc* に戻されます。

SQLColAttributes() は、将来拡張することはできますが、 SQLDescribeCol() よりも、各列ごとに同じ情報を入手するのに呼び出さなければならない回数が多くなります。

fDescType 記述子タイプがデータベース・サーバーで用いられないものである場合、その記述子の想定結果に応じて、 *rgbDesc* に空ストリングか、または *pfDesc* にゼロが戻されます。

列は、番号で識別され (1 から始めて左から右へ順次番号付けされる)、任意の順序で記述することができます。

fDescType を SQL_DESC_COUNT に設定して SQLColAttributes() を呼び出す操作は、SQLNumResultCols() を呼び出して戻せる列があるかどうかを判別する場合と同じ操作になります。

結果セットが存在するかどうかを判別する場合は、先に SQLNumResultCols() を呼び出してから SQLColAttributes() を呼び出してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

診断

表 30. SQLColAttributes SQLSTATE

SQLSTATE	説明	解説
07009	列番号が無効	引き数 *icol* に指定された値が、 1 未満になっていました。
HY009	引き数値が無効	引き数 *fDescType* に指定した値は、 63 ページの表 29 に指定されている値に等しくありません。 引き数 *rgbDesc*、*pcbDesc*、または *pfDesc* は NULL ポインターです。
HY010	関数シーケンス・エラー	*hstmt* に対し、 SQLPrepare() または SQLExecDirect() より先に、この関数が呼び出されました。
HYC00	ドライバーでサポートされていない	列 *icol* の、データベース・サーバーから戻される SQL データ・タイプが DB2 UDB CLI で認識されません。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLColumnPrivileges - 表の列に関連した特権の入手

目的

SQLColumnPrivileges() は、指定された表について、列のリストおよび関連した特権を戻します。情報は SQL 結果セットに戻されますが、これは、照会で生成された結果セットの処理に使用するのと同じ関数を使って検索することができます。

構文

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

関数引き数

表 31. SQLColumnPrivileges 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*Statement Handle*	入力	ステートメント・ハンドル
SQLCHAR *	*CatalogName*	入力	3 分割の表名のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* の長さ。0 に設定してください。
SQLCHAR *	*SchemaName*	入力	表名のスキーマ修飾子。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* の長さ。
SQLCHAR *	*TableName*	入力	表名。
SQLSMALLINT	*NameLength3*	入力	*TableName* の長さ。
SQLCHAR *	*ColumnName*	入力	列名で結果セットを修飾する *pattern-value* が入るバッファー。
SQLSMALLINT	*NameLength4*	入力	*ColumnName* の長さ。

使用法

結果は、 69 ページの表 32 にリストされている列を含む標準結果セットとして戻されます。結果セットは、TABLE_CAT、TABLE_SCHEM、TABLE_NAME、COLUMN_NAME、および PRIVILEGE の順になります。複数の特権が、指定された列と関連がある場合、それぞれの特権は別々の行として戻されます。一般的なアプリケーションでは、列特権情報を判別するために、 SQLColumns() への呼び出し後にこの関数を呼び出すことができます。アプリケーションは、この関数への入力引き数として、 SQLColumns() 結果セットの TABLE_SCHEM、 TABLE_NAME、COLUMN_NAME 列内に戻される文字ストリングを使用する必要があります。

多くの場合、SQLColumnPrivileges() の呼び出しは、システム・カタログに対する複雑な (そのため、経費のかさむ) 照会にマップされるので、慎重に使用する必要があり、何回も呼び出さなくて済むように結果を保管しておかなければなりません。

カタログ関数結果セットの VARCHAR 列は、 SQL92 制限と一貫性があるように 128 という最大長属性で宣言されています。 DB2 名は 128 未満なので、アプリケーションは出力バッファー用に常に 128 文字 (およびヌル終止符) を取り分けておくか、あるいは SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を使用して SQLGetInfo() を呼び出して、接続されている DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、 TABLE_NAME、および COLUMN_NAME 列の実際の長さをそれぞれ判別することができます。

ColumnName 引き数は検索パターンを受け入れることに注意してください。

今後のリリースでは、新しい列が追加されたり、既存の列名が変更されたりする可能性はありますが、現行列の位置は変更されません。

表 32. SQLColumnPrivileges によって戻される列

列番号/列名	データ・タイプ	説明
TABLE_CAT	VARCHAR(128)	これは常に NULL です。
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME が入っているスキーマの名前。
TABLE_NAME	NULL 以外の VARCHAR(128)	表またはビューの名前。
COLUMN_NAME	NULL 以外の VARCHAR(128)	指定された表またはビューの列の名前。
GRANTOR	VARCHAR(128)	特権を付与したユーザーの許可 ID。
GRANTEE	VARCHAR(128)	特権が付与されるユーザーの許可 ID。
PRIVILEGE	VARCHAR(128)	列特権。次のいずれかになります。 • INSERT • REFERENCES • SELECT • UPDATE
IS_GRANTABLE	VARCHAR(3)	被認可者が他のユーザーに特権を付与することが許可されているかどうかを示します。 YES または NO のいずれか。

注: DB2 CLI で使われる列名は、 X/Open CLI CAE 仕様スタイルに準拠します。列のタイプ、内容、および順序は、 ODBC において SQLColumnPrivileges() の結果セット用に定義されているものと同じです。

列と関連のある複数の特権がある場合、それぞれの特権は、結果セット内に別の行として戻されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

診断

表 33. SQLColumnPrivileges SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、 SQL_NTS と等価ではありませんでした。
HY010	関数シーケンス・エラー	ステートメント・ハンドルのカーソルがオープンしています。 このステートメント・ハンドル用の接続がありません。

制約事項

なし

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);
```

参照

- 71 ページの『SQLColumns - 表の列情報の入手』
- 268 ページの『SQLTables - 表情報の取得』

SQLColumns - 表の列情報の入手

目的

SQLColumns() は、指定された表に列のリストを戻します。情報は SQL 結果セットに戻されますが、このセットは、SELECT ステートメントで生成された結果セットの取り出しに使用する関数と同じ関数で検索することができます。

構文

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

関数引き数

表 34. SQLColumns の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLCHAR *	*szCatalogName*	入力	結果セットを修飾する *pattern-value* を入れられるバッファー。 *Catalog* は、 3 つの部分で構成される表名の最初の部分です。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	*cbCatalogName*	入力	*szCatalogName* の長さ。 0 に設定してください。
SQLCHAR *	*szSchemaName*	入力	スキーマ名で結果セットを修飾する *pattern-value* を入れられるバッファー。
SQLSMALLINT	*cbSchemaName*	入力	*szSchemaName* の長さ。
SQLCHAR *	*szTableName*	入力	表名で結果セットを修飾する *pattern-value* を保管する可能性のあるバッファー。
SQLSMALLINT	*cbTableName*	入力	*szTableName* の長さ。
SQLCHAR *	*szColumnName*	入力	列名で結果セットを修飾する *pattern-value* を保管する可能性のあるバッファー。
SQLSMALLINT	*cbColumnName*	入力	*szColumnName* の長さ。

使用法

この関数は、表または表リストの列に関する情報を検索します。

標準の結果セットが、SQLColumns() から戻されます。結果セットの列は、 72 ページの表 35 にリストされています。今後のリリースでは、アプリケーションが REMARKS 列を越えてさらに列を追加できるようになる予定です。

SQLColumns

szCatalogName、*szSchemaName*、*szTableName*、および *szColumnName* の各引き数では、検索パターンが受け入れられます。ワイルドカード文字と一緒にエスケープ文字を指定して、検索パターン内で実際の文字が使われるようにすることができます。エスケープ文字は、SQL_ATTR_ESCAPE_CHAR 環境属性上に指定します。

この関数では、 SQLDescribeCol() または SQLColAttributes() で検索される、結果セットの列に関する情報は戻されません。結果セットの列情報をアプリケーション・プログラムで得たい場合は、効率を上げるため常に SQLDescribeCol() または SQLColAttributes() を呼び出すようにしてください。 SQLColumns() は、システム・カタログを対象とする複合照会にマップされますが、大量のシステム・リソースを必要とすることがあります。

表 35. SQLColumns によって戻される列

列名	データ・タイプ	説明
TABLE_CAT	VARCHAR(128)	現行サーバー。
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME が入っているスキーマの名前。
TABLE_NAME	VARCHAR(128)	表またはビューの名前。
COLUMN_NAME	VARCHAR(128)	列 ID。指定された表またはビューの列の名前。
DATA_TYPE	NULL 以外の SMALLINT	列の SQL データ・タイプを識別します。
TYPE_NAME	NULL 以外の VARCHAR(128)	DATA_TYPE に対応するデータ・タイプの名前を表す文字ストリング。
LENGTH_PRECISION	INTEGER	DATA TYPE が推定の数値データ・タイプである場合、この列には列の小数部精度のビット数が入れられます。厳密な数値データ・タイプである場合、この列には、列内で使用できる小数桁数の合計数が入れられます。時刻、タイム・スタンプのデータ・タイプの場合、この列には、秒の小数部分の精度の桁数が入れられます。その他の場合、この列は NULL になります。 **注:** 通常、精度の ODBC 定義は、データ・タイプを保管する桁数です。
BUFFER_LENGTH	INTEGER	SQLBindCol()、SQLGetData()、および SQLBindParam() の呼び出し時に SQL_DEFAULT が指定された場合は、この列からデータを保管するバイトの最大数。
NUM_SCALE	SMALLINT	列の位取り。位取りが該当しないデータ・タイプの場合は、 NULL が戻されます。

表 35. *SQLColumns* によって戻される列 (続き)

列名	データ・タイプ	説明
NUM_PREC_RADIX	SMALLINT	10 または 2 または NULL のどれか。 DATA TYPE が推定の数値データ・タイプである場合、この列には 2 が入れられ、LENGTH_PRECISION 列には、この列で許可されているビット数が入れられます。 DATA_TYPE が厳密なデータ・タイプである場合、この列には値 10 が入れられ、 LENGTH_PRECISION および NUM_SCALE の各列には、その列で許可されている 10 進数字の数が入れられます。 数値データ・タイプの場合、 DBMS から 10 または 2 の NUM_PREC_RADIX が戻されることがあります。 基数が該当しないデータ・タイプの場合は、NULL が戻されます。
NULLABLE	NULL 以外の SMALLINT	この列で NULL 値が受け入れられない場合は、SQL_NO_NULLS。 この列で NULL 値が受け入れられる場合は、SQL_NULLABLE になります。
REMARKS	VARCHAR(254)	この列に関する記述情報が入れられる場合があります。
COLUMN_DEF	VARCHAR(254)	この列のデフォルト値。デフォルト値が数値リテラルの場合、この列には単一引用符なしの数値リテラルの文字表示が入れられます。デフォルト値が文字ストリングである場合、この列は単一引用符で囲まれた当該ストリングになります。デフォルト値が DATE、TIME、および TIMESTAMP などの*疑似リテラル* である場合、この列の値は単一引用符なしの疑似リテラルのキーワード (CURRENT DATE など) になります。 デフォルト値として NULL が指定された場合は、この列から単一引用符なしのワード NULL が戻されます。デフォルト値を切り捨てなければ表示できない場合、この列の値は単一引用符なしの TRUNCATED になります。デフォルト値が指定されない場合、この列の値は NULL になります。
DATETIME_CODE	INTEGER	この列は現在 NULL になっています。
CHAR_OCTET_LENGTH	INTEGER	文字データ・タイプ列のオクテットの最大長になります。 1 バイト文字セットの場合、この値は LENGTH_PRECISION と同じになります。他のデータ・タイプの場合は、NULL になります。
ORDINAL_POSITION	NULL 以外の INTEGER	表の列の序数部。表の最初の列が 1 番になります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 36. SQLColumns SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、 SQL_NTS と等価ではありませんでした。
HY010	関数シーケンス・エラー	ステートメント・ハンドルのカーソルがオープンしています。 このステートメント・ハンドル用の接続がありません。

SQLConnect - データ・ソースへの接続

目的

SQLConnect() は、ターゲット・データベースへの接続を確立します。ターゲット SQL データベース、および任意指定で許可名、認証ストリングを、アプリケーション・プログラムから提供してください。

この関数より先に SQLAllocConnect() を呼び出す必要があります。

SQLAllocStmt() より先にこの関数を呼び出す必要があります。

構文

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

関数引き数

表 37. SQLConnect の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル
SQLCHAR *	*szDSN*	入力	データ・ソース: データベースの名前または別名。
SQLSMALLINT	*cbDSN*	入力	*szDSN* 引き数の内容の長さ
SQLCHAR *	*szUID*	入力	許可名 (ユーザー ID)
SQLSMALLINT	*cbUID*	入力	*szUID* 引き数の内容の長さ
SQLCHAR *	*szAuthStr*	入力	認証ストリング (パスワード)
SQLSMALLINT	*cbAuthStr*	入力	*szAuthStr* 引き数の内容の長さ

使用法

SQLSetConnectOption() を使用して、アプリケーション・プログラムのさまざまな接続特性 (オプション) を定義できます。

SQLConnect() への入力長さ引き数 (*cbDSN*、 *cbUID*、*cbAuthStr*) は、関連データの実際の長さに設定できます。この長さには NULL 終了文字は含まれません。関連データが NULL 終了になっていることを示すには SQL_NTS を実行します。

szDSN および *szUID* 引き数値の前または後に付けられたブランクは、引用符で囲まれていない限り処理前に取り除かれます。

接続を機能させるには、システム上で事前にデータ・ソースを定義しておく必要があります。 iSeries では、リレーショナル・データベース (RDB) ディレクトリー項目の処理 (WRKRDBDIRE) コマンドを使って、どのデータ・ソースがすでに定義済みかを判断することができ、またオプションで、さらに別のデータ・ソースを定義することもできます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 38. SQLConnect SQLSTATE

SQLSTATE	説明	解説
08001	データ・ソースに接続不可	ドライバーがデータ・ソース (サーバー) との接続を確立できませんでした。
08002	接続は使用中	指定した *hdbc* は、データ・ソースとの接続の確立に使用されたもので、その接続はまだオープンしたままです。
08004	データ・ソースが接続の確立を拒否	データ・ソース (サーバー) が接続の確立を拒否しました。
28000	許可指定が無効	引き数 *szUID* または *szAuthStr* に指定した値は、データ・ソースで定義されている制約事項に違反しています。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *cbDSN* に指定されている値は 0 未満でしたが、SQL_NTS と等価ではなく、引き数 *szDSN* は NULL ポインターになっていませんでした。 引き数 *cbUID* に指定されている値は 0 未満でしたが、SQL_NTS と等価ではなく、引き数 *szUID* は NULL ポインターになっていませんでした。 引き数 *cbAuthStr* に指定されている値は 0 未満でしたが、 SQL_NTS と等価ではなく、引き数 *szAuthStr* は NULL ポインターになっていませんでした。 左右が対応していない二重引用符 (") が、 *szDSN*、*szUID*、または *szAuthStr* 引き数で検出されました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。
HY501 *	データ・ソース名が無効	引き数 *szDSN* に指定したデータ・ソース名は無効です。

制約事項

IBM DBMS では、暗黙接続 (またはデフォルト・データベース) オプションはサポートされていません。SQL ステートメントを実行するには、先に `SQLConnect()` を呼び出す必要があります。 iSeries では、 1 つのジョブにおける同一データ・ソースへの複数の同時接続はサポートされません。

新規リリースで DB2 UDB CLI を使用すると、 SQLConnect() で SQL0144 メッセージが出されることがあります。 これは、データ・ソース (サーバー) が古い SQL パッケージをもっているので、削除する必要があることを示します。そのパッケージを削除するには、サーバー・システムで次のようなコマンドを実行します。

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

その後の SQLConnect() 呼び出しで、新しい SQL パッケージが作成されます。

例

31 ページの『例』SQLAllocEnv() を参照してください。

参照

- 27 ページの『SQLAllocConnect - 接続の割り振り』
- 35 ページの『SQLAllocStmt - ステートメント・ハンドルの割り振り』

SQLCopyDesc - 記述ステートメントのコピー

目的

SQLCopyDesc() は、ソース・ハンドルに関連したデータ構造のフィールドを、ターゲット・ハンドルに関連したデータ構造にコピーします。

ターゲット・ハンドルに関連したデータ構造にある既存データは上書きされますが、 ALLOC_TYPE フィールドは変更されません。

構文

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

関数引き数

表 39. SQLCancel の引き数

データ・タイプ	引き数	使用法	説明
SQLHDESC	*sDesc*	入力	ソース記述子ハンドル
SQLHDESC	*tDesc*	入力	ターゲット記述子ハンドル

使用法

自動生成行のハンドルとステートメントのパラメーター記述子は、 GetStmtAttr() を呼び出せば得られます。

戻りコード

- SQL_SUCCESS
- SQL_INVALID_HANDLE
- SQL_ERROR

SQLDataSources - データ・ソース・リストの入手

目的

SQLDataSources() は、使用可能なターゲット・データベースのリストを一度に 1 つずつ戻します。データベースは、使用可能なようにカタログされていなければなりません。カタログの詳細は、 SQLConnect() の使用法の注意事項を参照するか、またはリレーショナル・データベース (RDB) ディレクトリー項目の処理 (WRKRDBDIRE) コマンドのオンライン・ヘルプを参照してください。

通常、接続を確立する前に SQLDataSources() を呼び出して、接続先の使用可能なデータベースを判別します。

構文

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

関数引き数

表 40. SQLDataSources の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*EnvironmentHandle*	入力	環境ハンドル。
SQLSMALLINT	*Direction*	入力	リスト内の最初のデータ・ソース名か、またはその次のものの名前を要求するのにアプリケーション・プログラムが使います。 *Direction* は、次の値のみとることができます。 • SQL_FETCH_FIRST • SQL_FETCH_NEXT
SQLCHAR *	*ServerName*	出力	検索したデータ・ソース名を保管するバッファーを指すポインターです。
SQLSMALLINT	*BufferLength1*	入力	*ServerName* が指すバッファーの最大長。これは SQL_MAX_DSN_LENGTH + 1 より小か等しくなければなりません。
SQLSMALLINT *	*NameLength1Ptr*	出力	*ServerName* に戻す使用可能な最大バイト数を保管する場所へのポインター。
SQLCHAR *	*説明*	出力	データ・ソースの記述が戻される先のバッファーを指すポインター。 DB2 UDB CLI は、 DBMS に対してカタログされたデータベースに関連した***注釈*** 欄を戻します。
SQLSMALLINT	*BufferLength2*	入力	*Description* バッファーの最大長。
SQLSMALLINT *	*NameLength2Ptr*	出力	この関数が、データ・ソースの記述を戻すのに使用できる実際のバイト数を戻す場所へのポインター。

使用法

アプリケーション・プログラムは、 *Direction* を SQL_FETCH_FIRST または SQL_FETCH_NEXT に設定すれば、いつでもこの関数を呼び出すことができます。

SQL_FETCH_FIRST を指定すると、リスト内の最初のデータベースが常に戻されます。

SQL_FETCH_NEXT を指定すると、次のようになります。

- SQL_FETCH_FIRST 呼び出しの直後、リスト内の 2 番目のデータベースが戻されます。
- 他のどの SQLDataSources() 呼び出しよりも先に、リスト内の最初のデータベースが戻されます。
- リスト内にデータベースがなくなると、SQL_NO_DATA_FOUND が戻されます。この関数をもう一度呼び出すと、最初のデータベースが戻されます。
- その後は常に、リスト内の次のデータベースが戻されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

エラー状況

表 41. SQLDataSources SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	引き数 *ServerName* に戻されたデータ・ソース名は、引き数 *BufferLength1* に指定された値よりも長いです。引き数 *NameLength1Ptr* には、データ・ソース名全体の長さが入ります。(関数からは SQL_SUCCESS_WITH_INFO が戻されます。) 引き数 *Description* に戻されたデータ・ソース名は、引き数 *BufferLength2* に指定された値よりも長いです。引き数 *NameLength2Ptr* には、データ・ソース記述全体の長さが入ります。 (関数からは SQL_SUCCESS_WITH_INFO が戻されます。)
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY000	一般エラー	エラーが発生しましたが、そのエラーには特定の SQLSTATE はなく、特定の SQLSTATE も定義されていません。 SQLError() が引き数 *ErrorMsg* に戻すエラー・メッセージに、このエラーとその原因についての説明があります。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *ServerName*、*NameLength1Ptr*、 *Description*、または *NameLength2Ptr* は NULL ポインターです。 Direction の値が無効です。
HY013	予想外のメモリー処理エラー	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーにアクセスできません。

表 41. SQLDataSources SQLSTATE (続き)

SQLSTATE	説明	解説
HY103	Direction オプションが範囲外	引き数 *Direction* に指定した値は、 SQL_FETCH_FIRST または SQL_FETCH_NEXT に等しくありません。

許可

なし。

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 ) ;


}
```

参照

なし。

SQLDescribeCol - 列属性の記述

目的

SQLDescribeCol() は、 SELECT ステートメントで生成された結果セットの指定列の結果記述情報 (列名、タイプ、精度) を戻します。

アプリケーション・プログラムで、記述子情報のうちの 1 つの属性だけが必要な場合、 SQLDescribeCol() の代わりに SQLColAttributes() 関数を使用することもできます。詳細については、63 ページの『SQLColAttributes - 列属性』を参照してください。

この関数の前に、 SQLPrepare() と SQLExecDirect() のどちらかを呼び出す必要があります。

この関数 (または SQLColAttributes()) は、通常 SQLBindCol() よりも先に呼び出されます。

構文

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

関数引き数

表 42. SQLDescribeCol の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*icol*	入力	記述される列番号
SQLCHAR *	*szColName*	出力	列名バッファーへのポインター
SQLSMALLINT	*cbColNameMax*	入力	*szColName* バッファーのサイズ
SQLSMALLINT *	*pcbColName*	出力	*szColName* 引き数に戻せるバイト数。*pcbColName* が *cbColNameMax* より大か等しい場合、列名 (*szColName*) は *cbColNameMax - 1* に切り捨てられます。
SQLSMALLINT *	*pfSqlType*	出力	列の SQL データ・タイプ
SQLINTEGER *	*pcbColDef*	出力	データベースに定義されている列の精度。 *fSqlType* に図形 SQL データ・タイプが指示されている場合、この変数は列に入れられる 2 バイト文字 の最大数を示します。
SQLSMALLINT *	*pibScale*	出力	データベースに定義されている列の位取り (SQL_DECIMAL、 SQL_NUMERIC、SQL_TIMESTAMP にのみ適用可)。

表 42. SQLDescribeCol の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT *	*pfNullable*	出力	NULLS がこの列で認められるかどうかを指示します。 • SQL_NO_NULLS • SQL_NULLABLE

使用法

列は数値で識別されますが、番号は 1 から始めて左から右へ順次付けられます。また列は、任意の順序で記述される場合があります。

有効なポインターとバッファー・スペースを、 *szColName* 引き数で使用可能にする必要があります。他のポインター引き数のどれかに NULL ポインターが指定されると、 DB2 UDB CLI はアプリケーション・プログラムにはこの情報は必要ないと見なすので、何も戻ってきません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

`SQLDescribeCol()` から SQL_ERROR または SQL_SUCCESS_WITH_INFO のどちらかが戻される場合は、`SQLError()` 関数を呼び出して、以下の SQLSTATE のいずれかを得ることができます。

表 43. SQLDescribeCol SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	引き数 *szColName* に戻された列名が、引き数 *cbColNameMax* に指定される値よりも長くなっています。引き数 *pcbColName* の値は、列名全体の長さになります。 (関数からは SQL_SUCCESS_WITH_INFO が戻されます。)
07005 *****	SELECT ステートメントではありません。	*hstmt* に関連するステートメントから結果セットが戻されませんでした。記述するための列がありません。 (まず、`SQLNumResultCols()` を呼び出して、結果セットの行があるかどうか判別してください。)
07009	列番号が無効	引き数 *icol* に指定された値が、1 未満になっていました。 引き数 *icol* に指定された値が、結果セットの列の数より大きくなっています。
40003 *****	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 43. SQLDescribeCol SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *cbColNameMax* に指定されている長さが、 1 未満でした。 引き数 *szColName* または *pcbColName* が NULL ポインターです。
HY010	関数シーケンス・エラー	*hstmt* に対し、 SQLPrepare() または SQLExecDirect() より先に、この関数が呼び出されました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。
HYC00	ドライバーでサポートされていない	*icol* 列の SQL データ・タイプが、 DB2 UDB CLI で認識されませんでした。

例

以下の例の完全なリストについては、 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』を参照してください。

```
/************************************************************************
** 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
```

参照

- 63 ページの『SQLColAttributes - 列属性』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 203 ページの『SQLNumResultCols - 結果列の数の取得』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLDescribeParam - パラメーター・マーカーの記述を戻す

目的

SQLDescribeParam() は、作成された SQL ステートメントに関連したパラメーター・マーカーの記述を戻します。この情報は、実装パラメーター記述子 (IPD) のフィールドから入手することもできます。

構文

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

関数引き数

表 44. SQLDescribeParam の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLSMALLINT	ParameterNumber	入力	1 から始めて、パラメーターの昇順に順に付けられたパラメーター・マーカー番号。
SQLSMALLINT *	DataTypePtr	出力	パラメーターの SQL データ・タイプを戻す先のバッファーを指すポインター。
SQLINTEGER *	ParameterSizePtr	出力	データ・ソースで定義されているとおりの、対応するパラメーター・マーカーの列サイズまたは式を戻す先のバッファーを指すポインター。
SQLSMALLINT *	DecimalDigitsPtr	出力	データ・ソースで定義されているとおりの、対応するパラメーター・マーカーの列または式の小数桁数を戻す先のバッファーを指すポインター。
SQLSMALLINT *	NullablePtr	出力	パラメーターに NULL 値を使用できるかどうかを示す値を戻す先のバッファーを指すポインター。この値は、IPD の SQL_DESC_NULLABLE フィールドから読み取られます。 次のいずれかです。 • SQL_NO_NULLS - パラメーターには NULL 値を使えません (これがデフォルト値)。 • SQL_NULLABLE - パラメーターに NULL 値を使えます。 • SQL_NULLABLE_UNKNOWN - パラメーターに NULL 値を使用できるかどうか判別できません。

使用法

パラメーター・マーカーには、パラメーターの昇順に番号が付けられます。番号は、1 から始まって、SQL ステートメント内に出現する順序に準じます。

SQLDescribeParam

SQLDescribeParam() は、 SQL ステートメント内のパラメーターのタイプ (入力、出力、または入出力) を戻しません。プロシージャーの呼び出しの場合を除き、 SQL ステートメント内のすべてのパラメーターは入力パラメーターです。プロシージャーの呼び出し内の各パラメーターのタイプを判別するには、アプリケーション・プログラムで SQLProcedureColumns() を呼び出します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 45. SQLDescribeParam SQLSTATE

SQLSTATE	説明	解説
01000	警告	通知メッセージです。 (関数からは SQL_SUCCESS_WITH_INFO が戻されます。)
07009	記述子索引が無効	引き数 *ParameterNumber* に指定された値が、1 未満です。 引き数 *ParameterNumber* に指定された値は、関連した SQL ステートメント内のパラメーター数より多いです。 パラメーター・マーカーは、非 DML ステートメントの一部です。 パラメーター・マーカーは、選択リストの一部です。
08S01	通信リンク障害	関数の処理が完了する前に、 DB2 UDB CLI とその接続先のデータ・ソースの間の通信リンクに障害が起こりました。
21S01	挿入値リストが列リストに不一致	INSERT ステートメント内のパラメーター数が、そのステートメントに指定されている表内の列数と一致しません。
HY000	一般エラー	
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作取り消し	
HY009	引き数値が無効	引き数 *DataTypePtr*、*ParameterSizePtr*、*DecimalDigitsPtr*、または *NullablePtr* が NULL ポインターです。
HY010	関数シーケンス・エラー	*StatementHandle* の SQLPrepare() または SQLExecDirect() より先に、この関数が呼び出されました。

表 45. SQLDescribeParam SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予想外のメモリー処理エラー	メモリー不足状態という推定原因で、基礎を成すメモリー・オブジェクトにアクセスできないため、関数呼び出しを処理できません。

制約事項

なし。

参照

- 48 ページの『SQLBindParam - パラメーター・マーカーに対するバッファーのバインド』
- 61 ページの『SQLCancel - ステートメントの取り消し』
- 103 ページの『SQLExecute - ステートメントの実行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLDisconnect - データ・ソースからの切断

目的

SQLDisconnect() は、データベース接続ハンドルと関連する接続をクローズします。

この関数を呼び出した後、別のデータベースに接続する場合は、 SQLConnect() を呼び出し、そうしない場合は SQLFreeConnect() を呼び出してください。

構文

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

関数引き数

表 46. SQLDisconnect の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル

使用法

接続と関連するすべてのステートメント・ハンドルが解放される前に、アプリケーション・プログラムから SQLDisconnect が呼び出された場合、これらのハンドルは、 DB2 UDB CLI とデータベース間の接続切断処理が正常に実行された後で DB2 UDB CLI により解放されます。

SQL_SUCCESS_WITH_INFO が戻された場合は、データベースとの切断が正常実行されても、追加のエラーまたは実装固有の情報は利用可能であることが示唆されています。以下に例を示します。

- 切断処理後のクリーンアップ時に問題が発生した。
- アプリケーション・プログラムに依存しない事象 (通信障害など) が発生したため、現在の接続がない。

SQLDisconnect() 呼び出しが正常実行された後で、アプリケーション・プログラムで *hdbc* を再使用して、もう 1 回 SQLConnect() 要求を出すことができます。

hdbc が DUOW 2 フェーズ・コミット接続に参加している場合は、切断が即時には実行されない場合があります。実際の切断処理は、分散トランザクションに次回コミットが出されたときに実行されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 47. SQLDisconnect SQLSTATE

SQLSTATE	説明	解説
01002	切断エラー	切断中にエラーが発生しました。ただし、切断処理は正常に実行されました。 (関数からは SQL_SUCCESS_WITH_INFO が戻されます。)
08003	接続がオープンしていない	引き数 *hdbc* に指定されている切断はオープンしていませんでした。
25000	トランザクション状態が無効	引き数 *hdbc* で指定されている接続上に、処理中のトランザクションがあります。トランザクションが活動状態のままなので、接続の切断処理を実行できません。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

例

31 ページの『例』SQLAllocEnv() を参照してください。

参照

- 27 ページの『SQLAllocConnect - 接続の割り振り』
- 75 ページの『SQLConnect - データ・ソースへの接続』
- 271 ページの『SQLTransact - トランザクション管理』

SQLDriverConnect - (拡張) データ・ソースへの接続

目的

SQLDriverConnect() は SQLConnect() の代替関数です。どちらの関数も、ターゲット・データベースへの接続を確立しますが、 SQLDriverConnect() は、接続ストリングを使って、データ・ソース名、ユーザー ID、およびパスワードを判別します。これらの関数は同一であり、互換性の理由でサポートされています。

構文

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

関数引き数

表 48. SQLDriverConnect の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル
SQLHWND	*hwindow*	入力	ウィンドウ・ハンドル (プラットフォームに依存します)。 Windows では、これは親 Windows ハンドルです。 OS/2 では、これは親 PM ウィンドウ・ハンドルです。 AIX® では、これは親 MOTIF ウィンドウ・ハンドルです。 iSeries では、これは無視されます。
SQLCHAR *	*InConnectionString*	入力	完全、部分的、または空の (NULL ポインター) 接続ストリング (この後の構文と説明を参照)。
SQLSMALLINT	*StringLength1*	入力	*InConnectionString* の長さ。
SQLCHAR *	*OutConnectionString*	出力	完了した接続ストリング用のバッファーを指すポインター。 接続の確立が正常に完了した場合、このバッファーには、完了した接続ストリングが入っています。
SQLSMALLINT	*BufferLength*	入力	*OutConnectionString* が指すバッファーの最大サイズ。
SQLSMALLINT *	*StringLength2Ptr*	出力	*OutConnectionString* バッファーに戻すのに使用できるバイト数を指すポインター。 *StringLength2Ptr* の値が *BufferLength* より大きいか等しい場合、 *OutConnectionString* で完了した接続ストリングは、 *BufferLength* - 1 バイトに切り捨てられます。

表 48. *SQLDriverConnect* の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*DriverCompletion*	入力	DB2 UDB CLI が、いつ詳細情報をユーザーにプロンプトで要求すればよいかを示します。 指定できる値は次のとおりです。 • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE_REQUIRED • SQL_DRIVER_NOPROMPT

使用法

接続ストリングは、その接続を確立するのに必要な 1 つ以上の値を渡すのに使います。接続ストリングの内容と、*DriverCompletion* の値で、その接続の確立法が決まります。

接続ストリングの構文

上記のキーワードはいずれも、次のものに等しい属性をもっています。

DSN データ・ソースの名前。データベースの名前または別名。データ・ソース名が必要なのは、*DriverCompletion* が SQL_DRIVER_NOPROMPT である場合です。

UID 許可名 (ユーザー ID)

PWD 許可名に対応するパスワード。ユーザー ID 用のパスワードがない場合、空を指定します (PWD=;)。

現在 iSeries には DB2 UDB CLI 定義のキーワードはありません。

DriverCompletion の値は有効であることと検証されますが、すべて同じ動作が生じます。接続ストリングに入っている情報への接続が試みられます。十分な情報がないと、SQL_ERROR が戻されます。

接続を確立し終わったら、完了接続ストリングが戻されます。アプリケーション・プログラムが、特定のユーザー ID で同じデータベースに複数の接続を設定する必要のある場合、この出力接続ストリングを保管しておかなければなりません。保管しておけば、その後の `SQLDriverConnect()` 呼び出しで、このストリングを入力接続値として使うことができます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA_FOUND
- SQL_INVALID_HANDLE
- SQL_ERROR

エラー状況

ここでも、 75 ページの『SQLConnect - データ・ソースへの接続』で生成されるすべての診断を戻すことができます。下の表は、戻すことのできるその他の診断を示しています。

表 49. SQLDriverConnect SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	バッファー *szConnstrOut* は、接続ストリング全体を保管するのに十分な大きさではありません。引き数 *StringLength2Ptr* に、戻すのに使用できる接続ストリングの実際の長さが入っています。 (関数からは SQL_SUCCESS_WITH_INFO が戻されます。)
01S00	接続ストリング属性が無効	無効なキーワードまたは属性値が入力接続ストリングに指定されましたが、次のような措置のいずれかがとられたため、データ・ソースへの接続はとりあえず正常に完了しました。 • 未認識のキーワードは無視された。 • 無効な属性値は無視され、その代わりにデフォルト値が使われた。 (関数からは SQL_SUCCESS_WITH_INFO が戻されます。)
HY009	引き数値が無効	引き数 *InConnectionString*、*OutConnectionString*、または *StringLength2PTR* は NULL ポインターです。 引き数 DriverCompletion は 1 に等しくありません。
HY090	ストリングまたはバッファー長が無効	*StringLength1* に指定された値は 0 未満でしたが、 SQL_NTS に等しくありません。 *BufferLength* に指定された値は 0 未満です。
HY110	ドライバー完了が無効	引き数 *fCompletion* に指定した値は、有効値のどれにも等しくありません。

制約事項

なし。

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);
|      }
|  }
```

参照

- 75 ページの『SQLConnect - データ・ソースへの接続』

SQLEndTran - トランザクションのコミットまたはロールバック

目的

SQLEndTran() は、接続中の現在のトランザクションをコミットまたはロールバックします。

接続時点か、または SQLEndTran() の前回の呼び出し時点のどちらか後のほう以後にこの接続で実行されたすべてのデータベース変更がコミットまたはロールバックされます。

トランザクションが接続上で活動状態になっている場合に、アプリケーション・プログラムは、データベースとの接続を切断するには、まず SQLEndTran() を呼び出す必要があります。

構文

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

関数引き数

表 50. SQLEndTran の引き数

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*hType*	入力	ハンドルのタイプ。 SQL_HANDLE_ENV または SQL_HANDLE_DBC になっている必要があります。
SQLINTEGER	*ハンドル*	入力	COMMIT または ROLLBACK を実行する際に使用するハンドル。
SQLSMALLINT	*fType*	入力	トランザクションへの希望するアクション。この引き数の値は、以下のいずれかである必要があります。 • SQL_COMMIT • SQL_ROLLBACK • SQL_COMMIT_HOLD • SQL_ROLLBACK_HOLD • SQL_SAVEPOINT_NAME_ROLLBACK • SQL_SAVEPOINT_NAME_RELEASE

使用法

SQL_COMMIT または SQL_ROLLBACK でトランザクションを完了すると、次のような結果を生じます。

- SQLEndTran() の呼び出しの後もステートメント・ハンドルは有効のままになります。
- カーソル名、バインド・パラメーター、および列バインドは、トランザクション完了後も有効のままになります。
- オープン・カーソルはクローズされ、検索保留になっている結果セットはすべて廃棄されます。

SQL_COMMIT_HOLD または SQL_ROLLBACK_HOLD でトランザクションを完了しても、データベースの変更はやはりコミットまたはロールバックされますが、カーソルがクローズされることはありません。

接続上に現在活動状態のトランザクションが存在しない場合は、 SQLEndTran() を呼び出してもデータベース・サーバーへの効果はなく、 SQL_SUCCESS が戻されます。

COMMIT または ROLLBACK の実行中は、接続がないため、 SQLEndTran() は失敗することがあります。この場合、 COMMIT または ROLLBACK が処理されているかどうかはアプリケーション・プログラムには分からないので、データベース管理者に問い合わせる必要があるかもしれません。トランザクション・ログとその他のトランザクション管理作業の詳細については、 DBMS 製品情報を参照してください。

SQL_SAVEPOINT_NAME_ROLLBACK と SQL_SAVEPOINT_NAME_RELEASE のどちらかを使用する場合は、事前に SQLSetConnectAttr を使用して保管点の名前を設定しておく必要があります。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 51. SQLEndTran SQLSTATE

SQLSTATE	説明	解説
08003	接続がオープンしていない	*hdbc* は接続状態になっていません。
08007	トランザクション時に接続障害が発生	この関数の実行時に *hdbc* 関連の接続が失敗し、この障害の発生前に要求された COMMIT または ROLLBACK が実行されたかどうかが判別できません。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	SQL_SAVEPOINT_NAME_ROLLBACK または SQL_SAVEPOINT_NAME_RELEASE が使用されましたが、属性 SQL_ATTR_SAVEPOINT_NAME のために SQLSetConnectAttr() を呼び出すことによって、保管点の名前の確立が行われていません。
HY012	トランザクションの操作状態が無効	引き数 *fType* に指定された値が、 SQL_COMMIT にも SQL_ROLLBACK にもなっていませんでした。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

SQLError - エラー情報の検索

目的

DB2 UDB CLI 関数の最新の呼び出しに関連して、特定のステートメント、接続ハンドル、または環境ハンドルに関する診断情報が SQLError() から戻されました。

この情報は、標準化された SQLSTATE、固有のエラー・コード、およびテキスト・メッセージで構成されています。詳細については、16 ページの『DB2 UDB CLI アプリケーションでの診断』を参照してください。

別の関数呼び出しからの SQL_ERROR または SQL_SUCCESS_WITH_INFO の戻りコードを受け取った後、 SQLError() を呼び出してください。

構文

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

関数引き数

表 52. SQLError の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル。環境に関連する診断情報を表示するには、有効な環境ハンドルを渡します。 *hdbc* および *hstmt* を、それぞれ SQL_NULL_HDBC および SQL_NULL_HSTMT に設定してください。
SQLHDBC	*hdbc*	入力	データベース接続ハンドル接続に関連する診断情報を表示するには、有効な接続ハンドルを渡し、 *hstmt* を SQL_NULL_HSTMT に設定してください。 *henv* 引き数は無視されます。
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル。ステートメントに関連する診断情報を表示するには、有効なステートメント・ハンドルを渡してください。 *henv* および *hdbc* 引き数は無視されます。

表 52. SQLError の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*szSqlState*	出力	NULL 文字で切り捨てられた 5 文字のストリングで構成される SQLSTATE 先頭の 2 文字はエラー・クラスを、それに続く 3 文字はサブクラスを表します。これらの値は、 IBM 独自の SQLSTATE 値と製品独自の SQLSTATE 値で増幅されていますが、 X/Open SQL CAE 仕様および ODBC 仕様に定義されている SQLSTATE 値に直接対応しています。
SQLINTEGER *	*pfNativeError*	出力	固有のエラー・コード。 DB2 UDB CLI の場合、 *pfNativeError* 引き数値は DBMS から戻される SQLCODE 値になっています。 DBMS ではなく DB2 UDB CLI によってエラーが生成される場合、このフィールドは -99999 に設定されます。
SQLCHAR *	*szErrorMsg*	出力	実装定義のメッセージ・テキストを保管するバッファーへのポインター。 DB2 UDB CLI の場合は DBMS 生成のメッセージだけが戻され、 DB2 UDB CLI 自体からは問題を説明するメッセージ・テキストは戻されません。
SQLSMALLINT	*cbErrorMsgMax*	入力	バッファー *szErrorMsg* の最大 (割り振りの) 長。割り振る長さの推奨値は、 SQL_MAX_MESSAGE_LENGTH + 1 です。
SQLSMALLINT *	*pcbErrorMsg*	出力	*szErrorMsg* バッファーに戻せる合計バイト数を指すポインター。

使用法

SQLSTATE は、 IBM 独自の SQLSTATE 値と製品独自の SQLSTATE 値で増幅されていますが、 X/Open SQL CAE 仕様および X/Open SQL CLI スナップショットで定義された値です。

以下のものに関連する診断情報を表示するには、以下の操作を実行してください。

- 環境。有効な環境ハンドルを渡す。 *hdbc* および *hstmt* を、それぞれ SQL_NULL_HDBC および SQL_NULL_HSTMT に設定してください。
- 接続。有効な接続ハンドルを渡し、 *hstmt* を SQL_NULL_HSTMT に設定してください。 *henv* 引き数は無視されます。
- ステートメントに関連する診断情報を表示するには、有効なステートメント・ハンドルを渡してください。 *henv* および *hdbc* 引き数は無視されます。

同じハンドルを使って `SQLError()` 以外の関数を呼び出す場合は、先に、1 つの DB2 UDB CLI 関数によって生成された診断情報を取り出さないと、直前の関数呼び出しに関する情報は失われます。これは、診断情報が 2 回目の DB2 UDB CLI 関数呼び出しで生成されたものかどうかに関係なくあてはまります。

エラー・メッセージが切り捨てられないようにするには、 SQL_MAX_MESSAGE_LENGTH + 1 のバッファー長を宣言してください。メッセージ・テキストがこの長さより長くなることはありません。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND
- SQL_SUCCESS

診断

SQLError() がそれ自体の診断情報を生成することはないので、 SQLSTATE は定義されません。引き数 szSqlState、pfNativeError、szErrorMsg、または pcbErrorMsg が NULL ポインターであった場合、SQL_ERROR が戻されます。

例

以下の例の完全なリストについては、 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』を参照してください。

```
/************************************************************************
** 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];
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 - ステートメントの直接実行

目的

SQLExecDirect は、指定された SQL ステートメントを直接実行します。このステートメントを実行できるのは、1 回だけです。また、接続されたデータベース・サーバーはこのステートメントを準備できる必要があります。

構文

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

関数引き数

表 53. SQLExecDirect の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル。 *hstmt* に関連するオープン・カーソルは無効です。詳細については、127 ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』を参照してください。
SQLCHAR *	*szSqlStr*	入力	SQL ステートメント・ストリング。接続されたデータベース・サーバーはこのステートメントを準備できる必要があります。
SQLINTEGER	*cbSqlStr*	入力	*szSqlStr* 引き数の内容の長さ。この長さは、ステートメントの正確な長さ、またはステートメントが NULL 文字で終了している場合は SQL_NTS のどちらかに設定する必要があります。

使用法

SQL ステートメントは、COMMIT または ROLLBACK できません。 COMMIT または ROLLBACK を発行するには、 SQLTransact() を呼び出してください。サポートされている SQL ステートメントの詳細については 4 ページの表 1 を参照してください。

SQL ステートメントの値としては、パラメーター・マーカーも有効です。パラメーター・マーカーは、SQL ステートメントでは ″?″ 文字で表示され、 SQLExecDirect() の呼び出し時にアプリケーション・プログラム変数値に置換するステートメント内の桁位置を表します。 SQLBindParam() は、アプリケーション・プログラム変数をそれぞれのパラメーター・マーカーにバインド (または関連付け) し、データ転送時に実行する必要のあるデータ変換があるかどうかを示します。 SQLExecDirect() を呼び出す前に、すべてのパラメーターをバインドしてください。

SQL ステートメントが SELECT の場合は、 SQLExecDirect() によりカーソル名が生成され、カーソルがオープンされます。アプリケーション・プログラムで SQLSetCursorName() を使用してカーソル名とステートメント・ハンドルを関連付けた場合、 DB2 UDB CLI はこのアプリケーション・プログラム生成のカーソル名を内部生成のカーソル名と関連付けます。

SQLExecDirect

SELECT ステートメントにより生成された結果セットの行を検索するには、 SQLExecDirect() が正常に戻された後で SQLFetch() を呼び出してください。

SQL ステートメントが位置の決まった DELETE または位置の決まった UPDATE である場合、ステートメントが参照するカーソルは、行に置かれます。さらに、 SQL ステートメントは同じ接続ハンドルで別のステートメント・ハンドルに定義される必要があります。

ステートメント・ハンドルではオープン・カーソルは無効です。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL ステートメントが検索 UPDATE または検索 DELETE で、検索条件に合う行がない場合は、SQL_NO_DATA_FOUND が戻されます。

診断

表 54. SQLExecDirect SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *szSqlStr* が NULL ポインターになっていました。 引き数 *cbSqlStr* は 1 未満でしたが、 SQL_NTS と同じになっていませんでした。
HY010	関数シーケンス・エラー	このステートメント・ハンドルに接続がないか、またはオープン・カーソルがあります。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

注: このステートメントの実行時に、 DBMS で生成される SQLSTATE 値は他にも多くあります。

例

109 ページの『例』SQLFetch() を参照してください。

参照

- 103 ページの『SQLExecute - ステートメントの実行』
- 108 ページの『SQLFetch - 次のデータ行』
- 251 ページの『SQLSetParam - パラメーターの設定』

SQLExecute - ステートメントの実行

目的

SQLExecute() は、 SQLPrepare() で正常に準備作成されたステートメントを 1 回または複数回実行します。このステートメントは、 SQLBindParam() でパラメーター・マーカーにバインドされたアプリケーション・プログラム変数の現在値を使用して実行されます。

構文

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

関数引き数

表 55. SQLExecute の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル。 hstmt に関連するオープン・カーソルは無効です。詳細については、 127 ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』を参照してください。

使用法

SQL ステートメントの値としては、パラメーター・マーカーも有効です。パラメーター・マーカーは、SQL ステートメントでは ″?″ 文字で表示され、 SQLExecute() の呼び出し時にアプリケーション・プログラム変数値に置換するステートメント内の桁位置を表します。 SQLBindParam() は、アプリケーション・プログラム変数をそれぞれのパラメーター・マーカーにバインド (または関連付け) し、データ転送時に実行する必要のあるデータ変換があるかどうかを示します。 SQLExecute() を呼び出す前に、すべてのパラメーターをバインドしてください。

SQLExecute() 呼び出し結果の処理が終われば、アプリケーション・プログラムで新規の (または同じ) アプリケーション・プログラム変数値を指定してこのステートメントを再実行できるようになっています。

SQLExecDirect() で実行されたステートメントを SQLExecute() を呼び出して再実行することはできません。最初に SQLPrepare() を呼び出す必要があります。

SQL ステートメントが SELECT の場合は、 SQLExecute() がカーソル名を生成し、カーソルをオープンします。アプリケーション・プログラムで SQLSetCursorName() を使用してカーソル名をステートメント・ハンドルを関連付けた場合、 DB2 UDB CLI はこのアプリケーション・プログラム生成のカーソル名を内部生成のカーソル名と関連付けます。

SELECT ステートメントを 2 回以上実行するには、アプリケーション・プログラムで SQL_CLOSE オプションを指定して SQLFreeStmt() を呼び出し、カーソルをクローズします。 SQLExecute() 時のステートメント・ハンドルではオープン・カーソルは無効です。

SELECT ステートメントにより生成された結果セットの行を検索するには、 SQLExecute() が正常に戻された後で SQLFetch() を呼び出してください。

SQLExecute

SQL ステートメントが位置の決まった DELETE または位置の決まった UPDATE である場合、ステートメントが参照するカーソルは、 SQLExecute() の呼び出し時に行に置かれ、同じ接続ハンドルで別のステートメント・ハンドルに定義される必要があります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL ステートメントが検索 UPDATE または検索 DELETE で、検索条件に合う行がない場合は、SQL_NO_DATA_FOUND が戻されます。

診断

SQLExecute() の SQLSTATE には、 **HY**009 を除いて SQLExecDirect() のすべての SQLSTATE が含まれており (102 ページの表 54 参照)、以下の表に示す SQLSTATE も追加されます。

表 56. SQLExecute SQLSTATE

SQLSTATE	説明	解説
HY010	関数シーケンス・エラー	指定された *hstmt* が準備作成状態になっていませんでした。先に SQLPrepare を呼び出さないで、SQLExecute() を呼び出しました。
注: このステートメントの実行時に、 DBMS で生成される SQLSTATE 値は他にも多くあります。		

例

210 ページの『例』SQLPrepare() を参照してください。

参照

- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 209 ページの『SQLPrepare - ステートメントの準備作成』
- 108 ページの『SQLFetch - 次のデータ行』
- 251 ページの『SQLSetParam - パラメーターの設定』

SQLExtendedFetch - 行配列の取り出し

目的

SQLExtendedFetch() は、各バインド列ごとに、複数の行の入ったデータ・ブロック (*rowset (行セット)* と呼びます) を戻すことで、 SQLFetch() の機能を拡張します。行セットのサイズは、SQLSetStmtAttr() 呼び出し上の SQL_ROWSET_SIZE 属性で決定します。

アプリケーション・プログラムは、一度に 1 つのデータ行を取り出すには、 SQLFetch() を呼び出さなければなりません。

構文

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

関数引き数

表 57. SQLExtendedFetch の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLSMALLINT	FetchOrientation	入力	取り出しの方向。有効値については、 114 ページの表 62 を参照してください。
SQLINTEGER	FetchOffset	入力	相対的位置づけのための行オフセット。
SQLINTEGER *	RowCountPtr	出力	実際に取り出す行数。処理時にエラーが起きた場合、*RowCountPtr* が、そのエラーの起きた行の前にある行 (行セット内の) の序数部を指します。最初の行の取り出しでエラーが起きた場合、 *RowCountPtr* は値 0 を指します。
SQLSMALLINT *	RowStatusArray	出力	状況値の配列。要素数は、行セット内の行数に等しくなければなりません (SQL_ROWSET_SIZE 属性で定義されているとおり)。次のように、取り出された各行の状況値が戻されます。 • SQL_ROW_SUCCESS 取り出された行数が、状況配列内の要素数より少ない (つまり、行セットのサイズより小さい) 場合、残りの状況要素は SQL_ROW_NOROW に設定されます。 DB2 UDB CLI では、取り出しの開始以後に、行が更新または削除されたかどうかを検出できません。したがって、次に示す ODBC 定義の状況値は示されません。 • SQL_ROW_DELETED • SQL_ROW_UPDATED

使用法

SQLExtendedFetch() は、行セットの配列の取り出しを行うのに使います。アプリケーション・プログラムは、 SQL_ROWSET_SIZE 属性を指定して SQLSetStmtAttr() を呼び出して配列のサイズを指定します。

SQLExtendedFetch() の最初の呼び出しの前、カーソルは第 1 行の前に置かれています。
SQLExtendedFetch() の呼び出しの後、カーソルは、取り出したばかりの行セット内の最後の行要素に対応する結果セット内の行上に置かれています。

DB2 UDB CLI は、 SQLBindCol() 関数でバインドされた結果セット内のすべての列を対象に、必要に応じてバインド列のデータを変換し、その列にバインドされている場所にそのデータを保管します。結果セットは、行に準じた方法でバインドしなければなりません。つまり、第 1 行内のすべての列の値は連続していて、その後に 2 行目が続き、その後同様に続くことを意味します。また、標識変数を使用すると、その変数はすべて 1 つの連続保管位置に戻されます。

この手順を使って複数の行を取り出す場合、すべての列はバインドされている必要があり、また、ストレージは連続していなければなりません。この関数を使って SQL プロシージャーの結果セットから行を取り出す場合、 SQL_FETCH_NEXT の方向だけがサポートされます。 SQL_ROWSET_SIZE に指定した行数に十分なストレージを割り振るのは、ユーザーの責任です。

SQLExtendedFetch() で SQL_FETCH_NEXT 以外の方向を使用する場合、カーソルはスクロール可能カーソルでなければなりません。 SQL_ATTR_CURSOR_SCROLLABLE 属性設定の詳細については、 252 ページの『SQLSetStmtAttr - ステートメント属性の設定』を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

エラー状況

表 58. SQLExtendedFetch SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数値 *RowCountPtr* または *RowStatusArray* は NULL ポインターです。 引き数 FetchOrientation に指定された値は認識されていません。
HY010	関数シーケンス・エラー	SQLFetch() の呼び出しから、 SQL_CLOSE を指定した SQLFreeStmt() の呼び出しまでの間に、 StatementHandle 用の SQLExtendedFetch() を呼び出しました。 *StatementHandle* の SQLPrepare() または SQLExecDirect() より先に、この関数が呼び出されました。 data-at-execute (SQLParamData()、 SQLPutData()) の操作中に関数を呼び出しました。

制約事項

なし。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 103 ページの『SQLExecute - ステートメントの実行』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 108 ページの『SQLFetch - 次のデータ行』

SQLFetch - 次のデータ行

目的

SQLFetch() は、結果セットの次の行にカーソルを進め、バインド列を検索します。

SQLFetch() を使って、 SQLBindCol() で指定した変数内にデータを直接受信することができますが、 SQLGetData() を呼び出して、取り出し後の列を 1 つずつ受信することもできます。また、列バインド時に変換が指示されている場合は、 SQLFetch() の呼び出し時にデータ変換も実行されます。

構文

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

関数引き数

表 59. SQLFetch の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル

使用法

SQLFetch() を呼び出せるのは、 *hstmt* で実行された最新のステートメントが SELECT の場合のみです。

SQLBindCol() でバインドされたアプリケーション変数の数が、結果セットの列の数より多いと、 SQLFetch() が失敗します。

列バインド時に SQLBindCol() が呼び出されないと、 SQLFetch() を実行してもデータはアプリケーション・プログラムに戻されず、カーソルが次の行に進むだけになります。この場合は、SQLGetData() を呼び出して、すべての列を個々に得ることができます。アンバインド列のデータは、 SQLFetch() によりカーソルが次の行に進められた時点で廃棄されます。

バインド変数が小さくて SQLFetch() の戻りデータが入らない場合、データは切り捨てられます。文字データが切り捨てられると、SQL_SUCCESS_WITH_INFO が戻され、切り捨てを通知する SQLSTATE が生成されます。 SQLBindCol() の遅延出力引き数 *pcbValue* には、サーバーで検索される列データの実際の長さが指定されます。アプリケーション・プログラムでは、この出力長さを入力長さと比較 (SQLBindCol() の *pcbValue* および *cbValueMax* 引き数) し、切り捨てられた文字カラムを判別します。

10 進小数点の右側の桁が切り捨てられた場合、数値データ・タイプの切り捨ては報告されません。 10 進小数点の左側の桁が切り捨てられると、エラーが戻されます (診断の項を参照)。

図形データ・タイプの切り捨ては、文字データ・タイプと同じ方法で処理されます。ただし、*rgbValue* バッファーが、 SQLBindCol() に指定されている *cbValueMax* より小さいか等しい 2 バイトの倍数に最も近い値で満たされることを除きます。 DB2 UDB CLI とアプリケーション・プログラム間で転送される図形データが NULL 文字で終了することはありません。

結果セットのすべての行の検索が完了したか、またはその他の行の検索が必要ない場合は、 SQLFreeStmt() を呼び出してカーソルをクローズし、その他のデータと関連リソースを廃棄してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

結果セットに行がない場合か、または直前の SQLFetch() 呼び出しにより結果セットのすべての行の取り出しが完了した場合は、 SQL_NO_DATA_FOUND が戻されます。

診断

表 60. SQLFetch SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	戻された 2 個以上の列のデータが切り捨てられました。ストリング値の右桁が切り捨てられます。 (エラーが発生しなければ SQL_SUCCESS_WITH_INFO が戻されます。)
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	指定された *hstmt* が実行状態になっていませんでした。先に SQLExecute または SQLExecDirect を呼び出さないで、この関数を呼び出しました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

例

コード例については、 viii ページの『コードの特記事項情報』を参照してください。

```
/************************************************************************
** 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
```

SQLFetch

```
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)
{
```

SQLFetch

```
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 */
```

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 103 ページの『SQLExecute - ステートメントの実行』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 130 ページの『SQLGetCol - 結果セットの行での 1 つの列の検索』
- 114 ページの『SQLFetchScroll - スクロール可能カーソルからの取り出し』

SQLFetchScroll - スクロール可能カーソルからの取り出し

目的

SQLFetchScroll() 要求された方向に基づいてカーソルの位置を決定し、バインド列を検索します。

SQLFetchScroll() で SQLBindCol() に指定する変数にデータを直接受信したり、 SQLGetData() を呼び出して取り出しの後で個々に列を受信したりすることもできます。また、列バインド時に変換が指示されている場合は、SQLFetchScroll() の呼び出し時にデータ変換も実行されます。

構文

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

関数引き数

表 61. SQLFetchScroll の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*fOrient*	入力	取り出しの方向。有効値については、表 62 を参照してください。
SQLINTEGER	*fOffset*	入力	相対的位置づけのための行オフセット。

使用法

SQLFetchScroll() を呼び出せるのは、 *hstmt* で実行された最新のステートメントが SELECT の場合だけです。

fOrient パラメーターが、どのデータの取り出しよりも前にカーソル位置を決定することを除き、SQLFetchScroll() は、 SQLFetch() に似た働きをします。 SQLFetchScroll() で SQL_FETCH_NEXT 以外の方向を使用する場合、カーソルはスクロール可能カーソルでなければなりません。SQL_ATTR_CURSOR_SCROLLABLE 属性設定の詳細については、 252 ページの『SQLSetStmtAttr - ステートメント属性の設定』を参照してください。

この関数を使って SQL プロシージャーの結果セットから行を取り出す場合、 SQL_FETCH_NEXT の方向だけがサポートされます。

表 62. ステートメント属性

fOrient	説明
SQL_FETCH_NEXT	現行カーソル位置の後の行に移動。
SQL_FETCH_FIRST	結果セットの先頭行に移動。
SQL_FETCH_LAST	結果セットの最終行に移動。
SQL_FETCH_PRIOR	現行カーソル位置の前の行に移動。

表 62. ステートメント属性 (続き)

fOrient	説明
SQL_FETCH_RELATIVE	*fOffset* が以下の値になっている場合、その意味は以下のとおりです。 • 正。指定した行数だけカーソルを進める。 • 負。指定した行数だけカーソルを後退させる。 • ゼロ。カーソルを移動しない。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

診断

表 63. *SQLFetchScroll SQLSTATE*

SQLSTATE	説明	解説
01004	データは切り捨てられる	戻された 2 個以上の列のデータが切り捨てられました。ストリング値の右桁が切り捨てられます。 (エラーが発生しなければ SQL_SUCCESS_WITH_INFO が戻されます。)
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	方向が無効です。
HY010	関数シーケンス・エラー	指定された *hstmt* が実行状態になっていませんでした。先に SQLExecute または SQLExecDirect を呼び出さないで、この関数を呼び出しました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 103 ページの『SQLExecute - ステートメントの実行』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 130 ページの『SQLGetCol - 結果セットの行での 1 つの列の検索』
- 108 ページの『SQLFetch - 次のデータ行』

SQLForeignKeys - 外部キー列リストの入手

目的

SQLForeignKeys() は、指定された表の外部キーに関する情報を戻します。情報は SQL 結果セットに戻されますが、これは、照会で生成された結果の取り出しに使用するのと同じ関数を使って処理することができます。

構文

```
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);
```

関数引き数

表 64. SQLForeignKeys の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLCHAR *	PKCatalogName	入力	基本キー表のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*PKCatalogName* の長さ。 0 に設定してください。
SQLCHAR *	PKSchemaName	入力	基本キー表のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*PKSchemaName* の長さ。
SQLCHAR *	PKTableName	入力	基本キーの入った表の名前。
SQLSMALLINT	NameLength3	入力	*PKTableName* の長さ。
SQLCHAR *	FKCatalogName	入力	外部キーの入った表のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	NameLength4	入力	*FKCatalogName* の長さ。 0 に設定してください。
SQLCHAR *	FKSchemaName	入力	外部キーの入った表のスキーマ修飾子。
SQLSMALLINT	NameLength5	入力	*FKSchemaName* の長さ。
SQLCHAR *	FKTableName	入力	外部キーの入った表の名前。
SQLSMALLINT	NameLength6	入力	*FKTableName* の長さ。

使用法

PKTableName に表名が入っていて、 *FKTableName* が空ストリングである場合、 SQLForeignKeys() は、指定された表の基本キーと、それを参照するすべての外部キー (他の表内の) の入っている結果セットを戻します。

FKTableName に表名が入っていて、 *PKTableName* が空ストリングである場合、 SQLForeignKeys() は、指定された表内のすべての外部キーと、そのキーが参照する基本キー (他の表内の) の入っている結果セットを戻します。

PKTableName と *FKTableName* のどちらにも表名が入っている場合、 SQLForeignKeys() は、*PKTableName* に指定されている表の基本キーを参照する *FKTableName* に指定されている表内の外部キーを戻します。これは、1 つ以内のキーでなければなりません。

表名に関連付けられたスキーマ修飾子引き数を指定しない場合のスキーマ名のデフォルト値は、現在有効になっている現行接続のものになります。

表 65 は、 SQLForeignKeys() 呼び出しで生成された結果セットの列を示しています。基本キーに関連付けられている外部キーを要求した場合、結果セットは FKTABLE_CAT、 FKTABLE_SCHEM、FKTABLE_NAME、および ORDINAL_POSITION の順になります。外部キーに関連付けられている基本キーを要求した場合、結果セットは PKTABLE_CAT、 PKTABLE_SCHEM、 PKTABLE_NAME、および ORDINAL_POSITION の順になります。

今後のリリースでは、新しい列が追加されたり、既存の列が変更されたりする可能性はありますが、現行列の位置は変更されません。

表 65. SQLForeignKeys によって戻される列

列番号/列名	データ・タイプ	説明
1 PKTABLE_CAT	VARCHAR(128)	現行サーバー。
2 PKTABLE_SCHEM	VARCHAR(128)	PKTABLE_NAME が入っているスキーマの名前。
3 PKTABLE_NAME	NULL 以外の VARCHAR(128)	基本キーの入った表の名前。
4 PKCOLUMN_NAME	NULL 以外の VARCHAR(128)	基本キーの列名。
5 FKTABLE_CAT	VARCHAR(128)	現行サーバー。
6 FKTABLE_SCHEM	VARCHAR(128)	FKTABLE_NAME が入っているスキーマの名前。
7 FKTABLE_NAME	NULL 以外の VARCHAR(128)	外部キーの入った表の名前。
8 FKCOLUMN_NAME	NULL 以外の VARCHAR(128)	外部キーの列名。
9 ORDINAL_POSITION	NULL 以外の SMALLINT	1 から始まる列の序数部。
10 UPDATE_RULE	SMALLINT	SQL 操作が UPDATE の場合に、外部キーに対して取る次のようなアクション。 • SQL_RESTRICT • SQL_NO_ACTION IBM DB2 DBMS の更新規則は常に RESTRICT または SQL_NO_ACTION です。ただし、ODBC アプリケーション・プログラムでは、 IBM 以外の RDBMS の場合は次のような UPDATE_RULE 値が検出されることがあります。 • SQL_CASCADE • SQL_SET_NULL

表 65. SQLForeignKeys によって戻される列 (続き)

列番号/列名	データ・タイプ	説明
11 DELETE_RULE	SMALLINT	SQL 操作が DELETE の場合に、外部キーに対して取る次のようなアクション。 • SQL_CASCADE • SQL_NO_ACTION • SQL_RESTRICT • SQL_SET_DEFAULT • SQL_SET_NULL
12 FK_NAME	VARCHAR(128)	外部キー ID。データ・ソースに対して該当しない場合は NULL。
13 PK_NAME	VARCHAR(128)	基本キー ID。データ・ソースに対して該当しない場合は NULL。

注: DB2 UDB CLI で使われる列名は、X/Open CLI CAE 仕様スタイルに準拠します。列のタイプ、内容、および順序は、 ODBC において `SQLForeignKeys()` の結果セット用に定義されているものと同じです。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 66. SQLForeignKeys SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルは、ステートメント・ハンドル上ですでにオープンしています。
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *PKTableName* と *FKTableName* はどちらも NULL ポインターです。
HY010	関数シーケンス・エラー	
HY014	ハンドルが不足	内部リソースに起因して DB2 UDB CLI はハンドルを割り振れませんでした。
HY090	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、 SQL_NTS と等価ではありませんでした。 表または所有者名の長さが、サーバーでサポートされる最大長より長いです。 159 ページの『SQLGetInfo - 一般情報の取得』を参照してください。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI では、表名の修飾子として *catalog* をサポートしていません。

表 66. SQLForeignKeys SQLSTATE (続き)

SQLSTATE	説明	解説
HYT00	タイムアウト満了	

制約事項

なし。

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);
        }
    }
```

参照

- 213 ページの『SQLPrimaryKeys - 表の基本キー列の入手』
- 262 ページの『SQLStatistics - 基本表の索引情報と統計情報の取得』

SQLFreeConnect - 接続ハンドルの解放

目的

SQLFreeConnect() は接続ハンドルを無効にし、解放します。接続ハンドルに関連するすべての DB2 UDB CLI リソースが解放されます。

この関数より先に SQLDisconnect() を呼び出す必要があります。

SQLFreeEnv() を呼び出してアプリケーション・プログラムの終了処理を続行するか、SQLAllocHandle() を呼び出して新規の接続ハンドルを割り振ります。

構文

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

関数引き数

表 67. SQLFreeConnect の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル

使用法

接続がまだ存在しているのにこの関数を呼び出すと、SQL_ERROR が戻され、接続ハンドルは有効のままになります。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 68. SQLFreeConnect SQLSTATE

SQLSTATE	説明	解説
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	*hdbc* に対し、SQLDisconnect() より先にこの関数が呼び出されました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

例

31 ページの『例』SQLAllocEnv() を参照してください。

参照

- 90 ページの『SQLDisconnect - データ・ソースからの切断』
- 123 ページの『SQLFreeEnv - 環境ハンドルの解放』

SQLFreeEnv - 環境ハンドルの解放

目的

SQLFreeEnv() は環境ハンドルを無効にし、解放します。環境ハンドルに関連したすべての DB2 UDB CLI リソースが解放されます。

この関数より先に SQLFreeConnect() を呼び出す必要があります。

この関数が、アプリケーション・プログラム終了処理の最後の DB2 UDB CLI ステップになります。

構文

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

関数引き数

表 69. SQLFreeEnv の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル

使用法

有効な接続ハンドルがまだ存在しているのにこの関数を呼び出すと、 SQL_ERROR が戻され、環境ハンドルは有効のままになります。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 70. SQLFreeEnv SQLSTATE

SQLSTATE	説明	解説
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	割り振りまたは接続状態になっている *hdbc* があります。SQLFreeEnv の前に、 *hdbc* に対して SQLDisconnect と SQLFreeConnect を呼び出してください。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

例

31 ページの『例』SQLAllocEnv() を参照してください。

SQLFreeEnv

参照

- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』

SQLFreeHandle - ハンドルの解放

目的

SQLFreeHandle() は、ハンドルを無効にし、解放します。

構文

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

関数引き数

表 71. SQLFreeHandle の引き数

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*hType*	入力	ハンドル・タイプ。 SQL_HANDLE_ENV、SQL_HANDLE_DBC、SQL_HANDLE_STMT、または SQL_HANDLE_DESC でなければなりません。
SQLINTEGER	*handle*	入力	解放するハンドル。

使用法

SQLFreeHandle() は、SQLFreeEnv()、 SQLFreeConnect()、および SQLFreeStmt() の機能を組み合わせたものです。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 72. SQLFreeHandle SQLSTATE

SQLSTATE	説明	解説
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	割り振りまたは接続状態になっている *hdbc* があります。SQLFreeHandle を呼び出す前に、 *hdbc* に対して SQLDisconnect と SQLFreeConnect を呼び出してください。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

SQLFreeHandle

参照

- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』
- 123 ページの『SQLFreeEnv - 環境ハンドルの解放』
- 127 ページの『SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)』

SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)

目的

SQLFreeStmt() は、ステートメント・ハンドルが参照するステートメントでの処理を終了します。この関数は、以下の場合に使用してください。

- カーソルをクローズする。
- パラメーターをリセットする。
- 変数から列をアンバインドする。
- ステートメント・ハンドルをドロップし、そのステートメント・ハンドルと関連する DB2 UDB CLI リソースを解放する。

SQLFreeStmt() は、 SQL ステートメントの実行および結果処理の後に呼び出されます。

構文

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

関数引き数

表 73. SQLFreeStmt の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*fOption*	入力	ステートメント・ハンドルを解放する方法を指定するオプション。このオプションの値は、以下のどれかに指定してください。 • SQL_CLOSE • SQL_DROP • SQL_UNBIND • SQL_RESET_PARAMS

使用法

SQLFreeStmt() は、以下のオプションで呼び出せます。

- SQL_CLOSE

  ステートメント・ハンドル (*hstmt*) と関連するカーソル (存在する場合) はクローズされ、保留中の結果は廃棄されます。アプリケーション・プログラムは、 *hstmt* にバインドされているアプリケーション・プログラム変数 (存在する場合) 内に、同じかまたは異なる値を指定して SQLExecute() を呼び出せば、カーソルを再オープンできます。カーソル名は、ステートメント・ハンドルがドロップされるか、次の SQLSetCursorName() の実行が正常に完了するまで保持されます。ステートメント・ハンドルに関連するカーソルがない場合、このオプションには効果がありません (警告もエラーも生成されません)。

- SQL_DROP

  入力ステートメント・ハンドルに関連する DB2 UDB CLI リソースが解放され、ハンドルは無効になります。 オープン・カーソル (存在する場合) はクローズされ、保留中の結果はすべて廃棄されます。

- SQL_UNBIND

SQLFreeStmt

このステートメント・ハンドルでの直前の SQLBindCol() 呼び出しでバインドされたすべての列が解放されます (アプリケーション・プログラム変数またはファイル参照と、結果セット列の関係は無効になります)。

- SQL_RESET_PARAMS

  このステートメント・ハンドルでの直前の SQLBindParam() 呼び出しで設定されたパラメーターが解放されます。アプリケーション・プログラム変数またはファイル参照と、このステートメント・ハンドルの SQL ステートメントのパラメーター・マーカー間の関係は無効になります。

直前に以下のステートメントを実行した場合に、ステートメント・ハンドルを再使用して、異なるステートメントを実行するには、以下のようにしてください。

- SELECT を実行した場合は、カーソルをクローズする。
- 異なる数またはタイプのパラメーターを使用した場合は、それらのパラメーターをリセットする。
- 異なる数またはタイプの列バインドを使用した場合は、それらの列をアンバインドする。

または、ステートメント・ハンドルをドロップして、新規のハンドルを割り振ることもできます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

fOption オプションが SQL_DROP に設定されていると、 SQLError() 呼び出し時に使用するステートメント・ハンドルがなくなるので、 SQL_SUCCESS_WITH_INFO は戻されません。

診断

表 74. SQLFreeStmt SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *fOption* に指定された値が、 SQL_CLOSE、SQL_DROP、SQL_UNBIND、または SQL_RESET_PARAMS のどれにもなっていませんでした。

例

109 ページの『例』SQLFetch() を参照してください。

参照

- 35 ページの『SQLAllocStmt - ステートメント・ハンドルの割り振り』
- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』

- 108 ページの『SQLFetch - 次のデータ行』
- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』
- 251 ページの『SQLSetParam - パラメーターの設定』

SQLGetCol - 結果セットの行での 1 つの列の検索

目的

SQLGetCol() は、結果セットの現在行の 1 つの列のデータを検索します。この関数は、 SQLFetch() への呼び出し時にデータをアプリケーション・プログラム変数に直接転送する SQLBindCol() の代わりに使用できます。また、SQLGetCol() は大規模な文字ベースのデータを断片的に検索する場合にも使用できます。

SQLFetch() は、 SQLGetCol() より先に呼び出す必要があります。

それぞれの列で SQLGetCol() を呼び出すと、 SQLFetch() が呼び出され、次の行を検索します。

構文

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

関数引き数

表 75. SQLGetCol の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*icol*	入力	要求されたデータ検索の対象の列番号。
SQLSMALLINT	*fCType*	入力	*icol* で識別される列のアプリケーション・データ・タイプ。以下のタイプがサポートされています。 • 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*	出力	検索された列データが保管されるバッファーへのポインター。

表 75. SQLGetCol の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*cbValueMax*	入力	*rgbValue* が指すバッファーの最大サイズ。*fcType* が SQL_DECIMAL または SQL_NUMERIC である場合、 *cbValueMax* は実際は精度と位取りでなければなりません。この 2 つの値を指定するには、(精度 * 256) + 位取りを使います。またこれは、SQLColAttributes() の使用時にこれらのデータ・タイプの長さとして戻される値でもあります。
SQLINTEGER *	*pcbValue*	出力	*rgbValue* バッファーに戻す際に DB2 UDB CLI が使用できるバイト数を示す値へのポインター。データが断片的に検索される場合、この値には、直前の SQLGetCol() の呼び出しで得た列データのすべてのバイトを除き、残っているバイト数も含まれます。 この列のデータ値が NULL の場合、この値は SQL_NULL_DATA になります。このポインターが NULL である場合に、 NULL データの入った列を SQLFetch() を使って取得すると、それを報告する手段がないため、この関数は失敗します。 SQLFetch() が図形データの入った列を取り出す場合、 *pcbValue* へのポインターは NULL であってはなりません。 NULL であると、この関数は失敗します。 *rgbValue* バッファーに取り入れられたデータの長さをアプリケーション・プログラムに通知する方法がないからです。

使用法

icol の値がバインド済みの列を指定していない限り、同じ行で SQLGetCol() を、 SQLBindCol() と一緒に使用することができます。一般的なステップは、以下のとおりです。

1. SQLFetch() - カーソルを先頭行に進め、先頭行を検索し、バインド列のデータを転送します。
2. SQLGetCol() - 指定された (アンバインド) 列のデータを転送します。
3. ステップ 2 をそれぞれ必要な行で繰り返します。
4. SQLFetch() - カーソルを次の行に進め、次の行を検索し、バインド列のデータを転送します。
5. 結果セットのそれぞれの行ごとにか、または結果セットがもう必要なくなるまで、ステップ 2、3、および 4 を繰り返します。

C データ・タイプ (*fCType*) が SQL_CHAR であるか、または *fCType* が SQL_DEFAULT であって、列タイプが CHAR または VARCHAR である場合、 SQLGetCol() は長列を検索します。

戻すのに使用できるデータが *cbValueMax* より大か等しい場合、 SQLGetCol() を呼び出すと、そのつど切り捨てが実行されます。データ切り捨てを示す SQLSTATE を伴った SQL_SUCCESS_WITH_INFO の関数

戻りコードは、切り捨てを表します。アプリケーション・プログラムは、同じ *icol* 値を指定した SQLGetCol() を再び呼び出して、切り捨て時点以降の同じアンバインド列のデータを後から得ることができます。アプリケーション・プログラムは、列全体を得るには、この関数で SQL_SUCCESS が戻されるまでこの呼び出しを繰り返します。次に SQLGetCol() を呼び出すと、 SQL_NO_DATA_FOUND が戻されます。

検索処理の途中ですべての列データ部分を廃棄するには、アプリケーション・プログラムで、該当する次の列位置に *icol* を設定して SQLGetCol() を呼び出します。行全体の未検索データを廃棄する場合は、アプリケーション・プログラムから SQLFetch() を呼び出して、カーソルを次の行に進めます。結果セットのデータがもう必要ない場合は、 SQLFreeStmt() を呼び出してカーソルをクローズしてください。

fCType 入力引き数は、 *rgbValue* が指すストレージに列データが入れられる前に実行する必要のあるデータ変換 (ある場合) のタイプを指定します。

SQL_ATTR_OUTPUT_NTS 属性の変更に SQLSetEnvAttr() は使用されなかった場合か、または、アプリケーション・プログラムが大きい塊に分かれたデータを取り出す場合は、 rgbValue に戻される内容は常に NULL 文字で終了します。アプリケーション・プログラムが、大きい塊に分かれたデータを取り出す場合、 NULL で終了するバイトは、そのデータの末尾部分にしか付け加えられません。

10 進小数点の右側の桁が切り捨てられた場合、数値データ・タイプの切り捨ては報告されません。 10 進小数点の左側の桁が切り捨てられると、エラーが戻されます (診断の項を参照)。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

前の SQLGetCol() 呼び出しで、この列のすべてのデータの検索が済んでいた場合、SQL_NO_DATA_FOUND が戻されます。

SQLGetCol() でゼロ長のストリングが検索されると、 SQL_SUCCESS が戻されます。 *pcbValue* には 0、*rgbValue* には NULL 終了文字が入ります。

前の SQLFetch() の呼び出しが失敗した場合、結果は未定義になっているので、SQLGetCol() を呼び出さないでください。

診断

表 76. SQLGetCol SQLSTATE

SQLSTATE	説明	解説
07006	制限付きデータ・タイプ属性違反	このデータ値は、引き数 *fCType* で指定されている C データ・タイプに変換できません。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 76. SQLGetCol SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *cbValueMax* に指定された値は 1 より小さく、*fCType* は SQL_CHAR です。 指定された列番号は、有効ではありません。 引き数 *rgbValue* または *pcbValue* は NULL ポインターです。
HY010	関数シーケンス・エラー	指定された *hstmt* が、カーソル位置状態になっていませんでした。先に SQLFetch() を呼び出さないで、この関数を呼び出しました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。
HYC00	ドライバーでサポートされていない	指定されたデータ・タイプの SQL データ・タイプは、認識されていますが、ドライバーではサポートされていません。 SQL データ・タイプからアプリケーション・データの *fCType* への変換が要求されましたが、ドライバーまたはデータ・ソースでは実行できません。

制約事項

ODBC の場合、同じステートメント・ハンドルの同じ行に対して、 SQLGetCol() で最後に検索された列よりも小さい番号の列を *icol* で指定しないようにする必要があります。また、ODBC では、 SQLGetCol() を使用して、最後のバインド列 (行の列でバインドされたものがある場合) より前に置かれた列のデータを検索することもできません。

DB2 UDB CLI では、これらの規則は緩和されています。つまり、 *icol* の値がバインド列を指定していない限り、任意の順序でバインド列より前に *icol* の値を指定することができます。

例

バインド列を使用する場合と SQLGetCol() を使用する場合の比較説明については、 109 ページの『例』 SQLFetch() を参照してください。

以下の例で使用されている check_error、initialize、および terminate 関数のリストについては、 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』を参照してください。

```
/************************************************************************
** 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
```

SQLGetCol

```
**        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 */
```

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 108 ページの『SQLFetch - 次のデータ行』

SQLGetConnectAttr - 接続属性の値の取得

目的

SQLGetConnectAttr() は、指定された接続オプションの現行設定を戻します。

これらのオプションは、 SQLSetConnectAttr() 関数で設定されます。

構文

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

関数引き数

表 77. SQLGetConnectAttr の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル
SQLINTEGER	*fAttr*	入力	検索する属性。詳細については、 233 ページの表 147 を参照してください。
SQLPOINTER	*pvParam*	出力	*fAttr* に関連する値。 *fAttr* の値に応じ、32 ビットの整数値、または NULL 終了文字ストリングへのポインターが有効です。
SQLINTEGER	*bLen*	入力	文字ストリングの場合、 pvParm に保管されるバイトの最大数。それ以外の場合は、使用されません。
SQLINTEGER *	*sLen*	出力	この属性が文字ストリングの場合、出力データの長さ。それ以外の場合は、使用されません。

使用法

SQLGetConnectAttr() が呼び出され、指定された *fAttr* が、 SQLSetConnectAttr を介して設定されておらず、デフォルトをもたない場合、 SQLGetConnectAttr() は SQL_NO_DATA_FOUND を戻します。

ステートメント・オプション設定は、 SQLGetConnectAttr() では検索できません。

診断

表 78. SQLGetConnectAttr SQLSTATE

SQLSTATE	説明	解説
08003	接続がオープンしていない	接続がオープンされている必要のある *fAttr* が指定されました。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 78. SQLGetConnectAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	属性タイプが範囲外	指定された *fAttr* 値は無効です。 引き数 *pvParam* は NULL ポインターです。
HYC00	ドライバーでサポートされていない	*fAttr* は、認識はされていますが、サポートされていません。

SQLGetConnectOption - 接続オプションの現行設定を戻す

目的

SQLGetConnectOption() は、指定された接続オプションの現行設定を戻します。

これらのオプションは、 SQLSetConnectOption() 関数で設定されます。

構文

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

関数引き数

表 79. SQLGetConnectOption の引き数

データ・タイプ	引き数	使用法	説明
HDBC	*hdbc*	入力	接続ハンドル
SQLSMALLINT	*fOption*	入力	検索するオプション。詳細については、 233 ページの表 147 を参照してください。
SQLPOINTER	*pvParam*	出力	*fOption* に関連する値。 *fOption* の値に応じ、32 ビットの整数値、または NULL 終了文字ストリングへのポインターが有効です。任意の戻り文字ストリングの最大長は、 SQL_MAX_OPTION_STRING_LENGTH バイト (NULL 終了バイトは除く) です。

使用法

SQLGetConnectOption() は、 SQLGetConnectAttr() と同じ関数を提供していますが、どちらの関数も互換性の理由でサポートされています。

SQLGetConnectOption() が呼び出され、指定された *fOption* が SQLSetConnectOption で設定されておらず、デフォルト値がない場合、 SQLGetConnectOption() は SQL_NO_DATA_FOUND を戻します。

ステートメント・オプション設定は、 SQLGetConnectOption() では検索できません。

診断

表 80. SQLGetConnectOption SQLSTATE

SQLSTATE	説明	解説
08003	接続がオープンしていない	オープン接続を求める *fOption* が指定されました。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	オプション・タイプが範囲外	指定された *fOption* 値は無効です。 引き数 *pvParam* は NULL ポインターです。
HYC00	ドライバーでサポートされていない	*fOption* は、認識はされていますが、サポートされていません。

SQLGetCursorName - カーソル名の取得

目的

SQLGetCursorName() は、入力ステートメント・ハンドルに関連したカーソル名を戻します。SQLSetCursorName() の呼び出しでカーソル名を明示的に設定していた場合、その名前が戻され、それ以外の場合は、暗黙生成された名前が戻されます。

構文

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

関数引き数

表 81. SQLGetCursorName の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLCHAR *	*szCursor*	出力	カーソル名
SQLSMALLINT	*cbCursorMax*	入力	バッファー *szCursor* の長さ。
SQLSMALLINT *	*pcbCursor*	出力	*szCursor* 引き数に戻せるバイト数。

使用法

名前が SQLSetCursorName() で設定された場合、または SELECT ステートメントがステートメント・ハンドルで実行された場合、 SQLGetCursorName() はカーソル名を戻します。どちらでもない場合、SQLGetCusorName() を呼び出すとエラーになります。

名前を SQLSetCursorName() で明示的に設定した場合、ステートメントがドロップされるか、または明示的に別の名前が設定されない限り、この名前が戻されます。

明示的に名前が設定されないと、 SELECT ステートメントの実行時に暗黙名が生成され、この名前が戻されます。暗黙カーソル名は、必ず SQLCUR で始まります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 82. SQLGetCursorName SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	*szCursor* に戻されたカーソル名が *cbCursorMax* の値より長いため、 *cbCursorMax* - 1 バイトの長さに切り捨てられます。引き数 *pcbCursor* の値は、戻すのに使用できるカーソル名全体の長さになります。この関数は SQL_SUCCESS_WITH_INFO を戻します。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *szCursor* または *pcbCursor* は NULL ポインターです。 引き数 *cbCursorMax* に指定された値が、1 未満になっていました。
HY010	関数シーケンス・エラー	ステートメント *hstmt* が実行状態になっていません。 SQLGetCursorName() を呼び出す前に、 SQLExecute()、 SQLExecDirect()、または SQLSetCursorName() を呼び出してください。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。
HY015	使用可能なカーソル名がない	*hstmt* にオープン・カーソルがなく、 SQLSetCursorName() で設定されたカーソル名がありません。 *hstmt* に関連するステートメントでは、カーソルの使用はサポートされていません。

制約事項

ODBC 生成のカーソル名は SQL_CUR で始まり、 X/Open CLI 生成のカーソル名は SQLCUR で始まります。 DB2 UDB CLI では SQLCUR を使用しています。

例

以下の例で使用されている check_error、initialize、および terminate 関数のリストについては、 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』を参照してください。

```
/*****************************************************************************
** 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 */
```

参照

- 103 ページの『SQLExecute - ステートメントの実行』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 240 ページの『SQLSetCursorName - カーソル名の設定』

SQLGetData - 列のデータの取得

目的

SQLGetData() は、結果セットの現在行の 1 つの列のデータを検索します。この関数は、 SQLFetch() への呼び出し時にデータをアプリケーション・プログラム変数に直接転送する SQLBindCol() の代わりに使用できます。また、 SQLGetData() は大規模な文字ベースのデータを断片的に検索する場合にも使用できます。

SQLFetch() は、 SQLGetData() より先に呼び出す必要があります。

それぞれの列で SQLGetData() を呼び出すと、 SQLFetch() が呼び出され、次の行を検索します。

SQLGetData() は、SQLGetCol() と同一であり、どちらの関数も互換性の理由でサポートされています。

構文

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

注: 適当なセクションの説明については、 130 ページの『SQLGetCol - 結果セットの行での 1 つの列の検索』を参照してください。

SQLGetDescField - 記述子フィールドの取得

目的

SQLGetDescField() は、記述子の値を取得します。 SQLGetDescField() は、 SQLGetDescRec() 関数を拡張した代替関数として使用できます。

この関数の機能は SQLDescribeCol() と類似していますが、 SQLGetDescField() はパラメーター記述子だけでなく行記述子からもデータを検索できるようになっています。

構文

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

関数引き数

表 83. SQLGetDescField の引き数

データ・タイプ	引き数	使用法	説明
SQLHDESC	*hdesc*	入力	記述子ハンドル
SQLSMALLINT	*irec*	入力	指定されたフィールドを検索するレコード番号。
SQLSMALLINT	*fDescType*	入力	表 84 を参照してください。
SQLPOINTER	*rgbDesc*	出力	バッファーへのポインター。
SQLINTEGER	*bLen*	入力	記述子バッファーの長さ (*rgbDesc*)
SQLINTEGER *	*sLen*	出力	記述子の中の実際に戻すバイト数。この引き数の値が *rgbDesc* バッファーの長さと等価またはそれより長くなっている場合、値は切り捨てられています。

表 84. fDescType 記述子タイプ

記述子	タイプ	説明
SQL_DESC_COUNT	SMALLINT	記述子のレコード数は、*rgbDesc* に戻されます。
SQL_DESC_ALLOC_TYPE	SMALLINT	記述子をアプリケーション・プログラムで明示的に割り振った場合は SQL_DESC_ALLOC_USER、実装で自動的に割り振った場合は SQL_DESC_ALLOC_AUTO。
SQL_DESC_NAME	CHAR(128)	*irec* の NAME フィールドを検索します。
SQL_DESC_TYPE	SMALLINT	*irec* の TYPE フィールドを検索します。

表 84. *fDescType* 記述子タイプ (続き)

記述子	タイプ	説明
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	SQL_DATETIME タイプのレコードの時間間隔コードを検索します。 SQL_DATETIME データ・タイプは、内部コードでさらに定義されています。コード値は、SQL_CODE_DATE、SQL_CODE_TIME、および SQL_CODE_TIMESTAMP です。
SQL_DESC_LENGTH	INTEGER	*irec* の LENGTH フィールドを検索します。
SQL_DESC_PRECISION	SMALLINT	*irec* の PRECISION フィールドを検索します。
SQL_DESC_SCALE	SMALLINT	*irec* の SCALE フィールドを検索します。
SQL_DESC_NULLABLE	SMALLINT	*irec* で NULL が有効である場合、 *rgbDesc* には SQL_NULLABLE が戻されます。その他の場合、 *rgbDesc* には SQL_NO_NULLS が戻されます。
SQL_DESC_UNNAMED	SMALLINT	これは、NAME フィールドが実際の名前である場合は SQL_NAMED ですが、 NAME フィールドが実装システム生成名である場合は SQL_UNNAMED です。
SQL_DESC_DATA_PTR	SQLPOINTER	*irec* のポインター・フィールドを検索します。
SQL_DESC_LENGTH_PTR	SQLPOINTER	*irec* の長さポインター・フィールドを検索します。
SQL_DESC_INDICATOR_PTR	SQLPOINTER	*irec* の標識ポインター・フィールドを検索します。

使用法

記述子のレコード数は、行記述子の場合は結果セットの列数、パラメーター記述子の場合はパラメーター数に対応します。

fDescType を SQL_DESC_COUNT に設定して `SQLGetDescField()` を呼び出す操作は、`SQLNumResultCols()` を呼び出して戻せる列があるかどうかを判別する場合と同じ操作になります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

診断

表 85. SQLGetDescField SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *fDescType* または *irec* に指定された値が有効ではありませんでした。 引き数 *rgbDesc* または *sLen* は NULL ポインターです。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLGetDescRec - 記述子レコードの取得

目的

SQLGetDescRec() は、記述子からレコード全体を取得します。SQLGetDescRec() は、 SQLDescField() 関数を簡潔化した代替関数として使用できます。

構文

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

関数引き数

表 86. SQLGetDescRec の引き数

データ・タイプ	引き数	使用法	説明
SQLHDESC	*hdesc*	入力	記述子ハンドル
SQLSMALLINT	*irec*	入力	情報を検索するレコード番号。
SQLCHAR *	*rgbDesc*	出力	レコードの NAME フィールド。
SQLSMALLINT	*cbDescMax*	入力	*rgbDesc* に保管するバイトの最大数。
SQLSMALLINT *	*pcbDesc*	出力	出力データの全長。
SQLSMALLINT *	*type*	出力	レコードの TYPE フィールド。
SQLSMALLINT *	*subtype*	出力	TYPE が SQL_DATETIME になっているレコードの場合は DATETIME_INTERVAL_CODE。
SQLINTEGER *	*length*	出力	レコードの LENGTH フィールド。
SQLSMALLINT *	*prec*	出力	レコードの PRECISION フィールド。
SQLSMALLINT *	*scale*	出力	レコードの SCALE フィールド。
SQLSMALLINT *	*nullable*	出力	レコードの NULLABLE フィールド。

使用法

SQLGetDescRec() を呼び出すと、 1 回の呼び出しで記述子レコードのすべてのデータが検索されます。記述子のレコード数を判別するにはやはり、 SQL_DESC_COUNT を指定して SQLGetDescField() を呼び出す必要があります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

診断

表 87. SQLGetDescRec SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *irec* に指定された値が有効ではありませんでした。 引き数 *rgbDesc*、*pcbDesc*、*type*,*subtype*、*length*、*prec*、*scale*、または *nullable* は NULL ポインターです。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLGetDiagField - 診断情報 (拡張可能) を戻す

目的

SQLGetDiagField() は、特定のステートメント、接続ハンドル、または環境ハンドルへの最新の呼び出しとして出された DB2 UDB CLI 関数に関連する診断情報を戻します。

この情報は、標準化された SQLSTATE、固有のエラー・コード、およびテキスト・メッセージで構成されています。詳細については、16 ページの『DB2 UDB CLI アプリケーションでの診断』を参照してください。

SQLGetDiagField() は、別の関数呼び出しから SQL_ERROR または SQL_SUCCESS_WITH_INFO の戻りコードを受信した後で呼び出すようにしてください。

注: データベース・サーバーによっては、ステートメント実行で SQL_NO_DATA_FOUND が戻されると、製品固有の診断情報が提供される場合もあります。

構文

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

関数引き数

表 88. SQLDiagField の引き数

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*hType*	入力	ハンドル・タイプ。
SQLINTEGER	*handle*	入力	診断情報が必要なハンドル。
SQLSMALLINT	*recNum*	入力	複数のエラーが出た場合に、どのエラーを検索すればよいかを示します。ヘッダー情報が要求された場合は、これは 0 でなければなりません。最初のエラー・レコードが 1 番になります。
SQLSMALLINT	*diagId*	入力	150 ページの表 89 を参照してください。
SQLPOINTER	*diagInfo*	出力	診断情報のバッファー。
SQLSMALLINT	*bLen*	入力	要求データが文字ストリングである場合は *diagInfo* の長さ。その他の場合は使用されません。
SQLSMALLINT *	*sLen*	出力	要求データが文字ストリングである場合は完全な診断情報の長さ。その他の場合は使用されません。

表 89. diagId タイプ

記述子	タイプ	説明
SQL_DIAG_RETURNCODE	SMALLINT	基礎となる関数の戻りコード。SQL_SUCCESS、SQL_SUCCESS_WITH_INFO、SQL_NO_DATA_FOUND、または SQL_ERROR が有効です。
SQL_DIAG_NUMBER	INTEGER	指定されたハンドルで使用可能な診断レコードの数。
SQL_DIAG_ROW_COUNT	INTEGER	ハンドルがステートメント・ハンドルの場合、指定されたハンドルの行の数。
SQL_DIAG_SQLSTATE	CHAR(5)	診断レコードに関連する 5 文字の SQLSTATE コード。 SQLSTATE コードは、移植可能な診断指示を備えています。
SQL_DIAG_NATIVE	INTEGER	診断レコードに関連する実装定義のエラー・コード。移植可能なアプリケーション・プログラムの場合は、この値をベースにした動作は無効です。
SQL_DIAG_MESSAGE_TEXT	CHAR(254)	診断レコードに関連する実装定義のメッセージ・テキスト。
SQL_DIAG_SERVER_NAME	CHAR(128)	接続を確立した SQLConnect() ステートメントで指定された、診断レコードに関連したサーバー名。

使用法

SQLSTATE は、 IBM 独自の SQLSTATE 値と製品独自の SQLSTATE 値で増幅されていますが、X/Open SQL CAE 仕様および X/Open SQL CLI スナップショットで定義された値です。

同じハンドルを使って SQLGetDiagField() 以外の関数を呼び出す場合は、先に、1 つの DB2 UDB CLI 関数によって生成された診断情報を取り出さないと、直前の関数呼び出しに関する情報は失われます。これは、診断情報が 2 回目の DB2 UDB CLI 関数呼び出しで生成されたものかどうかに関係なくあてはまります。

与えられた DB2 UDB CLI 関数呼び出しの後、複数の診断メッセージが使用可能になることがあります。SQLGetDiagField() を繰り返し呼び出して、これらのメッセージを一度に 1 つずつ検索することができます。 SQLGetDiagField() は、検索されるそれぞれのメッセージに SQL_SUCCESS を戻し、そのメッセージを使用可能なメッセージのリストから削除していきます。検索するメッセージがなくなると、SQL_NO_DATA_FOUND が戻されます。

特定のハンドルに保管される診断情報は、このハンドルを指定して SQLGetDiagField() を呼び出すか、または別の DB2 UDB CLI 関数を呼び出すと、クリアされます。 ただし、関連していても異なるハンドル・タイプを指定して SQLGetDiagField() を呼び出しても、与えられたハンドル・タイプに関連する情報はクリアされません。たとえば、接続ハンドルを入力して SQLGetDiagField() を呼び出しても、その接続のステートメント・ハンドルに関連するエラーはクリアされません。

エラー・メッセージのバッファー (*szDiagFieldMsg*) が短すぎる場合でも、 SQL_SUCCESS が戻されます。これは、 SQLGetDiagField() を再呼び出ししても、アプリケーション・プログラムで同じエラー・メッセージを検索することはできないためです。 *pcbDiagFieldMsg* には、メッセージ・テキストの実際の長さが戻されます。

エラー・メッセージが切り捨てられないようにするには、 SQL_MAX_MESSAGE_LENGTH + 1 のバッファー長を宣言してください。メッセージ・テキストがこの長さより長くなることはありません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

入力ハンドルに使用可能な診断情報がない場合、または SQLGetDiagField() を何度か呼び出してすべてのメッセージを検索し終わった場合は、 SQL_NO_DATA_FOUND が戻されます。

引き数 diagInfo または sLen が NULL ポインターであった場合、 SQL_ERROR が戻されます。

診断

SQLGetDiagField() がそれ自体の診断情報を生成することはないので、 SQLSTATE は定義されません。

制約事項

X/Open SQL CAE SQLSTATE は ODBC でも戻されますが、追加の IBM 定義の SQLSTATE が戻されるのは DB2 UDB CLI だけです。 ODBC ドライバー・マネージャーでも、標準値に加え SQLSTATE 値も戻されます。 ODBC 固有の SQLSTATE の詳細については、「*Microsoft ODBC Programmer's Reference*」を参照してください。

このため、依存関係は標準 SQLSTATE 値で構築するようにしてください。つまり、アプリケーション・プログラムでのブランチ・ロジックも標準 SQLSTATE にのみ依存することになります。デバッグの場合は、SQLSTATE 値を大きくして使用するのが最も実用的です。

SQLGetDiagRec - 診断情報 (短縮型) を戻す

目的

SQLGetDiagRec() は、特定のステートメント、接続ハンドルまたは環境ハンドルへの最新の呼び出しとして出された DB2 UDB CLI 関数に関連する診断情報を戻します。

この情報は、標準化された SQLSTATE、固有のエラー・コード、およびテキスト・メッセージで構成されています。詳細については、16 ページの『DB2 UDB CLI アプリケーションでの診断』を参照してください。

SQLGetDiagRec() は、別の関数呼び出しから SQL_ERROR または SQL_SUCCESS_WITH_INFO の戻りコードを受信した後で呼び出すようにしてください。

注: データベース・サーバーによっては、ステートメント実行で SQL_NO_DATA_FOUND が戻されると、製品固有の診断情報が提供される場合もあります。

構文

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

関数引き数

表 90. SQLGetDiagRec の引き数

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*hType*	入力	ハンドル・タイプ。
SQLINTEGER	*handle*	入力	診断情報が必要なハンドル。
SQLSMALLINT	*recNum*	入力	複数のエラーが出た場合に、どのエラーを検索すればよいかを示します。ヘッダー情報が要求された場合は、これは 0 でなければなりません。最初のエラー・レコードが 1 番になります。
SQLCHAR *	*szSqlState*	出力	NULL 文字で切り捨てられた 5 文字のストリングで構成される SQLSTATE 先頭の 2 文字はエラー・クラスを、それに続く 3 文字はサブクラスを表します。これらの値は、 IBM 独自の SQLSTATE 値と製品独自の SQLSTATE 値で増幅されていますが、 X/Open SQL CAE 仕様および ODBC 仕様に定義されている SQLSTATE 値に直接対応しています。

表 90. SQLGetDiagRec の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	*pfNativeError*	出力	固有のエラー・コード。 DB2 UDB CLI の場合、 *pfNativeError* 引き数値は DBMS から戻される SQLCODE 値になっています。 DBMS ではなく DB2 UDB CLI によってエラーが生成される場合、このフィールドは -99999 に設定されます。
SQLCHAR *	*szErrorMsg*	出力	実装定義のメッセージ・テキストを保管するバッファーへのポインター。 DB2 UDB CLI の場合は DBMS 生成のメッセージだけが戻され、 DB2 UDB CLI 自体からは問題を説明するメッセージ・テキストは戻されません。
SQLSMALLINT	*cbErrorMsgMax*	入力	バッファー *szErrorMsg* の最大 (割り振りの) 長。割り振る長さの推奨値は、 SQL_MAX_MESSAGE_LENGTH + 1 です。
SQLSMALLINT *	*pcbErrorMsg*	出力	*szErrorMsg* バッファーに戻せる合計バイト数を指すポインター。この数には、NULL 終了文字は含まれません。

使用法

SQLSTATE は、 IBM 独自の SQLSTATE 値と製品独自の SQLSTATE 値で増幅されていますが、 X/Open SQL CAE 仕様および X/Open SQL CLI スナップショットで定義された値です。

同じハンドルを使って SQLGetDiagRec() 以外の関数を呼び出す場合は、先に、1 つの DB2 UDB CLI 関数によって生成された診断情報を取り出さないと、直前の関数呼び出しに関する情報は失われます。これは、診断情報が 2 回目の DB2 UDB CLI 関数呼び出しで生成されたものかどうかに関係なくあてはまります。

与えられた DB2 UDB CLI 関数呼び出しの後、複数の診断メッセージが使用可能になることがあります。これらのメッセージは、 SQLGetDiagRec() を繰り返し呼び出して、一度に 1 つ検索できます。 SQLGetDiagRec() は、検索されるそれぞれのメッセージに SQL_SUCCESS を戻し、そのメッセージを使用可能なメッセージのリストから削除していきます。検索するメッセージがなくなると、 SQL_NO_DATA_FOUND が戻され、 SQLSTATE は "00000"、*pfNativeError* は 0 に設定され、 *pcbErrorMsg* および *szErrorMsg* は定義されません。

特定のハンドルで保管される診断情報は、このハンドルを指定して SQLGetDiagRec() か別の DB2 UDB CLI 関数が呼び出されると、クリアされます。 ただし、関連していても異なるハンドル・タイプを指定して SQLGetDiagRec() を呼び出しても、特定のハンドル・タイプに関連する情報はクリアされません。たとえば、接続ハンドルを入力して SQLGetDiagRec() を呼び出しても、その接続のステートメント・ハンドルに関連するエラーはクリアされません。

SQLGetDiagRec() を再呼び出ししても、アプリケーション・プログラムで同じエラー・メッセージを検索することはできないので、エラー・メッセージのバッファー (*szErrorMsg*) が短すぎる場合でも、 SQL_SUCCESS が戻されます。 *pcbErrorMsg* には、メッセージ・テキストの実際の長さが戻されます。

エラー・メッセージが切り捨てられないようにするには、 SQL_MAX_MESSAGE_LENGTH + 1 のバッファー長を宣言してください。メッセージ・テキストがこの長さより長くなることはありません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

入力ハンドルに使用可能な診断情報がない場合、または SQLGetDiagRec() を何度か呼び出してすべてのメッセージを検索し終わった場合は、 SQL_NO_DATA_FOUND が戻されます。

引き数 szSqlState、pfNativeError、szErrorMsg、または pcbErrorMsg が NULL ポインターであった場合、 SQL_ERROR が戻されます。

診断

SQLGetDiagRec() がそれ自体の診断情報を生成することはないので、 SQLSTATE は定義されません。

制約事項

X/Open SQL CAE SQLSTATE は ODBC でも戻されますが、追加の IBM 定義の SQLSTATE が戻されるのは DB2 UDB CLI だけです。 ODBC ドライバー・マネージャーでも、標準値に加え SQLSTATE 値も戻されます。 ODBC 固有の SQLSTATE の詳細については、「*Microsoft ODBC Programmer's Reference*」を参照してください。

このため、依存関係は標準 SQLSTATE 値で構築するようにしてください。つまり、アプリケーション・プログラムでのブランチ・ロジックも標準 SQLSTATE にのみ依存することになります。デバッグの場合は、SQLSTATE 値を大きくして使用するのが最も実用的です。

参照

- 149 ページの『SQLGetDiagField - 診断情報 (拡張可能) を戻す』

SQLGetEnvAttr - 環境属性の現行設定を戻す

目的

SQLGetEnvAttr() は、指定された環境属性の現行設定を戻します。

これらのオプションは、SQLSetEnvAttr() 関数で設定されます。

構文

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

関数引き数

表 91. SQLGetEnvAttr の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル
SQLINTEGER	*Attribute*	入力	検索する属性。詳細については、 246 ページの表 159 を参照してください。
SQLPOINTER	*Value*	出力	*Attribute* に関連する現行値。戻り値のタイプは *Attribute* に応じて異なります。
SQLINTEGER	*BufferLength*	入力	属性値が文字ストリングの場合は、*Value* が指すバッファーの最大サイズ。その他の場合は使用されません。
SQLINTEGER *	*StringLength*	出力	属性値が文字ストリングの場合は、出力データのバイト長。その他の場合は使用されません。

Attribute がストリングでない場合、 DB2 UDB CLI は *BufferLength* を無視し、 *StringLength* を設定しません。

使用法

SQLGetEnvAttr() は、環境ハンドルを割り振ってから解放するまでの間であればいつでも呼び出せます。この関数を使うと、環境属性の現行値が得られます。

診断

表 92. SQLGetEnvAttr SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	属性が範囲外	指定された *Attribute* 値は無効です。 引き数 *Value* または *StringLength* は NULL ポインターです。

SQLGetFunctions - 関数の取得

目的

SQLGetFunctions() は、特定の関数がサポートされているかどうかを照会します。このようにすると、異なるドライバーを使用していても、アプリケーション・プログラムをサポート・レベルの変化に適応させることができます。

この関数を呼び出す前に、SQLConnect() を呼び出し、データ・ソース (データベース・サーバー) への接続を確立する必要があります。

構文

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

関数引き数

表 93. SQLGetFunctions の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	データベース接続ハンドル。
SQLSMALLINT	*fFunction*	入力	照会中の関数。
SQLSMALLINT *	*pfSupported*	出力	照会中の関数がサポートされているかどうかに応じて、この関数が SQL_TRUE または SQL_FALSE を戻す場所へのポインター。

使用法

図 4 に、 *fFunction* 引き数の有効値とともに、それに対応する関数がサポートされているかどうかを示します。

注: アスタリスクの付いている値は、リモート・サーバーに接続されている場合はサポートされません。

```
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
```

図 4. サポートされている関数 (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
```

図 4. サポートされている関数 (2/2)

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 94. SQLGetFunctions SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *pfSupported* は NULL ポインターです。
HY010	関数シーケンス・エラー。まだ接続ハンドルを割り振ってはならない。	SQLConnect より先に SQLGetFunctions が呼び出されました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

SQLGetInfo - 一般情報の取得

目的

SQLGetInfo() は、アプリケーション・プログラムが現在接続されている DBMS に関する一般情報 (データ変換のサポートなど) を戻します。

構文

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

関数引き数

表 95. SQLGetInfo の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	データベース接続ハンドル。
SQLSMALLINT	*fInfoType*	入力	必要な情報のタイプ。
SQLPOINTER	*rgbInfoValue*	出力 (入力も可)	この関数が必要な情報を保管するバッファーへのポインター。検索される情報のタイプに応じ、戻される情報には 4 つのタイプがあります。 • 16 ビット整数値 • 32 ビット整数値 • 32 ビット 2 進値 • NULL 終了文字ストリング
SQLSMALLINT	*cbInfoValueMax*	入力	*rgbInfoValue* ポインターが指すバッファーの最大長。
SQLSMALLINT *	*pcbInfoValue*	出力	この関数が必要な情報を戻す場合の使用可能バイトの合計数を戻す場所へのポインター。 *pcbInfoValue* が指す場所の値が、*cbInfoValueMax* に指定されている *rgbInfoValue* バッファーのサイズより大きい場合、ストリング出力情報は *cbInfoValueMax* - 1 バイトに切り捨てられ、関数は SQL_SUCCESS_WITH_INFO で戻されます。

使用法

160 ページの表 96 に、 *fInfoType* の有効値、および SQLGetInfo() が戻す該当値の情報の説明をリストします。

SQLGetInfo

表 96. SQLGetInfo の戻り情報

fInfoType	形式	説明と注
SQL_ACTIVE_CONNECTIONS	短整数	1 回の適用でサポートされる活動状態の接続の最大数。 限界値がシステム・リソースによって異なることを示すゼロが戻されます。
SQL_ACTIVE_STATEMENTS	短整数	1 回の接続で有効な活動状態のステートメントの最大数。 限界値がシステム・リソースによって異なることを示すゼロが戻されます。
SQL_AGGREGATE_FUNCTIONS	32 ビット・マスク	集約関数のサポートを列挙しているビット・マスク: • SQL_AF_ALL • SQL_AF_AVG • SQL_AF_COUNT • SQL_AF_DISTINCT • SQL_AF_MAX • SQL_AF_MIN • SQL_AF_SUM
SQL_CATALOG_NAME	ストリング	文字ストリング ″Y″ は、サーバーがカタログ名をサポートしていることを示します。 ″N″ は、カタログ名がサポートされていないことを示します。
SQL_COLUMN_ALIAS	ストリング	接続が列の別名をサポートするかどうか。接続が列の別名の概念をサポートする場合は、値 ″Y″ が戻されます。

表 96. SQLGetInfo の戻り情報 (続き)

fInfoType	形式	説明と注
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 ビット・マスク	infoType に指定されているタイプのデータに対して CONVERT スカラー関数を使用した場合の、データ・ソースによってサポートされている変換を示します。ビット・マスクがゼロに等しい場合には、データ・ソースは、同じデータ・タイプへの変換を含め、指定されたタイプのデータのいかなる変換もサポートしません。 たとえば、データ・ソースが SQL_INTEGER データから SQL_DECIMAL データ・タイプへの変換をサポートするかどうかを調べるために、アプリケーションは SQL_CONVERT_INTEGER という finfoType を使って SQLGetInfo() を呼び出します。次に、アプリケーションは、戻されたビット・マスクと SQL_CVT_DECIMAL を AND 演算します。結果値がゼロでない場合には、変換がサポートされています。以下のビット・マスクは、サポートされている変換を判別するために使用されます。 • 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 ビット・マスク	ドライバーおよび関連したデータ・ソースでサポートされているスカラー変換関数を示します。 • SQL_FN_CVT_CONVERT - サポートされている変換関数を判別するために使用される。 • SQL_FN_CVT_CAST - サポートされているキャスト関数を判別するために使用される。

表 96. SQLGetInfo の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_CORRELATION_NAME	短整数	以下に、サーバーでサポートされている相関名の程度を示します。 • SQL_CN_ANY - サポートされており、任意の有効なユーザー定義の名前とすることができる。 • SQL_CN_NONE - 相関名がサポートされていない。 • SQL_CN_DIFFERENT - 相関名がサポートされているが、提示されている表名とは異なっていなければならない。
SQL_CURSOR_COMMIT_BEHAVIOR	16 ビット整数	COMMIT 操作によるカーソルへの影響を示します。値は、以下のとおりです。 • SQL_CB_DELETE - カーソルを破棄し、動的 SQL ステートメントのアクセス・プランを除去する。 • SQL_CB_CLOSE - カーソルを破棄するが、動的 SQL ステートメント (非照会ステートメントを含む) のアクセス・プランを保存する。 • SQL_CB_PRESERVE - 動的ステートメント (非照会ステートメントを含む) のカーソルおよびアクセス・プランを保存する。アプリケーションは引き続きデータを取り出すか、またはカーソルをクローズして、ステートメントを再び準備せずに照会を再実行することができる。 注: COMMIT の後、定位置更新または削除などのアクションをとる前に、カーソルを再び位置指定するために FETCH を発行する必要があります。
SQL_CURSOR_ROLLBACK_BEHAVIOR	16 ビット整数	ROLLBACK 操作によるカーソルへの影響を示します。値は、以下のとおりです。 • SQL_CB_DELETE - カーソルを破棄し、動的 SQL ステートメントのアクセス・プランを除去する。 • SQL_CB_CLOSE - カーソルを破棄するが、動的 SQL ステートメント (非照会ステートメントを含む) のアクセス・プランを保存する。 • SQL_CB_PRESERVE - 動的ステートメント (非照会ステートメントを含む) のカーソルおよびアクセス・プランを保存する。アプリケーションは引き続きデータを取り出すか、またはカーソルをクローズして、ステートメントを再び準備せずに照会を再実行することができる。 注: DB2 サーバーは、SQL_CB_PRESERVE プロパティーを持っていません。
SQL_DATA_SOURCE_NAME	ストリング	接続ハンドルの接続先のデータ・ソースの名前。

表 96. *SQLGetInfo* の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_DATA_SOURCE_READ_ONLY	ストリング	″Y″ という文字ストリングは、データベースが READ ONLY (読み取り専用) モードに設定されていることを示し、 ″N″ は、データベースが READ ONLY モードに設定されていないことを示します。
SQL_DBMS_NAME	ストリング	アクセス中の DBMS 製品の名前。 以下に例を示します。 • QSQ (DB2 UDB for iSeries の場合) • SQL (DB2 UDB (OS/2 版) の場合) • DSN (DB2 UDB for z/OS および OS/390 の場合)
SQL_DBMS_VER	ストリング	アクセス中の DBMS 製品のバージョン。

表 96. SQLGetInfo の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_DEFAULT_TXN_ISOLATION	32 ビット・マスク	サポートされているデフォルトのトランザクション分離レベル。 以下のマスクのいずれかが戻されます。 • SQL_TXN_READ_UNCOMMITTED - 変更は、すべてのトランザクションによって即座に認知される (ダーティー読み取り、反復不能読み取り、およびファントムが可能)。 これは UR レベルと等価である。 • SQL_TXN_READ_COMMITTED - トランザクション 1 によって読み取られる行を、トランザクション 2 によって変更したりコミットすることができる (反復不能読み取りおよびファントムが可能)。 これは CS レベルと等価である。 • SQL_TXN_REPEATABLE_READ - トランザクションは、検索条件または保留トランザクションと一致する行を追加または除去することができる (反復可能読み取り。しかしファントムが可能)。 これは RS レベルと等価である。 • SQL_TXN_SERIALIZABLE - 保留トランザクションによって影響されるデータは、他のトランザクションには使用不能である (反復可能読み取り。ファントムは不可)。 これは RR レベルと等価である。 • SQL_TXN_VERSIONING - IBM DBMS に適用できない。 • SQL_TXN_NOCOMMIT - 変更は正常操作の終わりに効率よくコミットされる。明示的コミットまたはロールバックは許可されていない。 これは DB2 UDB for iSeries 分離レベルである。 IBM 用語に言い換えると、以下のようになります。 • SQL_TXN_READ_UNCOMMITTED は非コミット読み取り (UR)。 • SQL_TXN_READ_COMMITTED はカーソル固定 (CS)。 • SQL_TXN_REPEATABLE_READ は読み取り固定 (RS)。 • SQL_TXN_SERIALIZABLE は反復可能読み取り (RR)。
SQL_DESCRIBE_PARAMETER	ストリング	パラメーターを記述できる場合には Y、パラメーターを記述できない場合には N。

表 96. *SQLGetInfo* の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_DRIVER_NAME	ストリング	データ・ソースにアクセスするために使用される、ドライバーのファイル名。
SQL_DRIVER_ODBC_VER	ストリング	ドライバーがサポートする ODBC のバージョン番号。 DB2 ODBC は 2.1 を戻す。
SQL_GROUP_BY	16 ビット整数	サーバーによる GROUP BY 文節のサポートの程度を示します。 • SQL_GB_NO_RELATION - GROUP BY 内の列と SELECT リスト内の列との間に関連はない。 • SQL_GB_NOT_SUPPORTED - GROUP BY はサポートされていない。 • SQL_GB_GROUP_BY_EQUALS_SELECT - GROUP BY は、選択リスト内にすべての非集約列を組み込んでいなければならない。 • SQL_GB_GROUP_BY_CONTAINS_SELECT - GROUP BY 文節は、 SELECT リスト内にすべての非集約列を含んでいなければならない。
SQL_IDENTIFIER_CASE	16 ビット整数	オブジェクト名 (表名など) の大文字小文字の区別を示します。 値は、以下のとおりです。 • SQL_IC_UPPER - 識別名がシステム・カタログ内に大文字で保管される。 • SQL_IC_LOWER - 識別名がシステム・カタログ内に小文字で保管される。 • SQL_IC_SENSITIVE - 識別名は大文字小文字の区別があり、システム・カタログ内に大文字小文字混合で保管される。 • SQL_IC_MIXED - 識別名は大文字小文字の区別がなく、システム・カタログ内に大文字小文字混合で保管される。 注: IBM DBMS での識別名には大文字小文字の区別がありません。
SQL_IDENTIFIER_QUOTE_CHAR	ストリング	引用符付きストリングの区切り文字として使用される文字。
SQL_LIKE_ESCAPE_CLAUSE	ストリング	LIKE 述部内でメタ文字パーセントおよび下線のためのエスケープ文字がサポートされているかどうかを示す文字ストリング。
SQL_MAX_CATALOG_NAME_LEN	16 ビット整数	カタログ修飾子名の最大長。3 つの部分の表名の最初の部分 (バイト単位)。
SQL_MAX_COLUMN_NAME_LEN	短整数	列名の最大長。
SQL_MAX_COLUMNS_IN_GROUP_BY	短整数	GROUP BY 文節中の列の最大数。
SQL_MAX_COLUMNS_IN_INDEX	短整数	SQL 索引中の列の最大数。
SQL_MAX_COLUMNS_IN_ORDER_BY	短整数	ORDER BY 文節中の列の最大数。
SQL_MAX_COLUMNS_IN_SELECT	短整数	SELECT ステートメント中の列の最大数。

表 96. SQLGetInfo の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_MAX_COLUMNS_IN_TABLE	短整数	SQL 表中の列の最大数。
SQL_MAX_CURSOR_NAME_LEN	短整数	カーソル名の最大長。
SQL_MAX_OWNER_NAME_LEN	短整数	所有者名の最大長。
SQL_MAX_ROW_SIZE	32 ビット符号なし整数	サーバーが基本表の単一行内でサポートする最大長 (バイト単位) を指定します。制限なしの場合にはゼロ。
SQL_MAX_SCHEMA_NAME_LEN	整数	スキーマ名の最大長。
SQL_MAX_STATEMENT_LEN	32 ビット符号なし整数	SQL ステートメント・ストリングの最大長 (バイト単位、ステートメントの中の空白文字の数を含む) を示します。
SQL_MAX_TABLE_NAME	短整数	表名の最大長。
SQL_MAX_TABLES_IN_SELECT	短整数	SELECT ステートメント中の表の最大数。
SQL_MULTIPLE_ACTIVE_TXN	ストリング	文字ストリング ″Y″ は、複数の接続上でトランザクションを活動状態にできることを示します。 ″N″ は、一度に 1 つの接続だけが、活動状態のトランザクションを持てることを示します。
SQL_NON_NULLABLE_COLUMNS	16 ビット整数	非ヌル可能列がサポートされているかどうかを示します。 • SQL_NNC_NON_NULL - 列を NOT NULL として定義できます。 • SQL_NNC_NULL - 列を NOT NULL として定義できません。

表 96. *SQLGetInfo* の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_NUMERIC_FUNCTIONS	32 ビット・マスク	サポートされているスカラー数字関数を示します。 サポートされる数字関数を決定するために、以下のビット・マスクが使用されます。 • 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 ビット整数	以下の ODBC 準拠のレベル: • SQL_OAC_NONE • SQL_OAC_LEVEL1 • SQL_OAC_LEVEL2
SQL_ODBC_SQL_CONFORMANCE	16 ビット整数	値は、以下のとおりです。 • SQL_OSC_MINIMUM - サポートされている最小 ODBC SQL グラマーを示します。 • SQL_OSC_CORE - サポートされている中核 ODBC SQL グラマーを示します。 • SQL_OSC_EXTENDED - サポートされている拡張 ODBC SQL グラマーを示します。 上記の 3 つのタイプの ODBC SQL グラマーの定義については、「*Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference*」 を参照してください。

表 96. SQLGetInfo の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_ORDER_BY_COLUMNS_IN_SELECT	ストリング	ORDER BY 文節内の列を選択リストに含める必要がある場合は ″Y″ に設定します。それ以外の場合は ″N″ に設定します。
SQL_OUTER_JOINS	ストリング	文字ストリング: • ″Y″ は、外部結合がサポートされていて、 DB2 ODBC が ODBC 外部結合要求構文をサポートしていることを示します。 • ″N″ は、これがサポートされていないことを示します。
SQL_OWNER_TERM または SQL_SCHEMA_TERM	ストリング	スキーマ (所有者) のデータベース・ベンダー用語。
SQL_OWNER_USAGE または SQL_SCHEMA_USAGE	32 ビット・マスク	実行されるときにスキーマ (所有者) と関連付けられる SQL ステートメントのタイプを示します。スキーマ修飾子 (所有者) は以下のとおりです。 • SQL_OU_DML_STATEMENTS - すべての DML ステートメントでサポートされます。 • SQL_OU_PROCEDURE_INVOCATION - プロシージャー呼び出しステートメントでサポートされます。 • SQL_OU_TABLE_DEFINITION - すべての表定義ステートメントでサポートされます。 • SQL_OU_INDEX_DEFINITION - すべての索引定義ステートメントでサポートされます。 • SQL_OU_PRIVILEGE_DEFINITION - すべての特権定義ステートメント (すなわち、 grant および revoke ステートメント) でサポートされます。
SQL_POSITIONED_STATEMENTS	32 ビット・マスク	定位置 UPDATE および定位置 DELETE ステートメントのサポートの度合いを示します。 • SQL_PS_POSITIONED_DELETE • SQL_PS_POSITIONED_UPDATE • SQL_PS_SELECT_FOR_UPDATE (サーバーがカーソルを介して列を更新できるようにするために、 <query expression> に FOR UPDATE 文節を指定する必要があるかどうかを示します。)
SQL_PROCEDURE_TERM	ストリング	プロシージャー用のデータ・ソース名。
SQL_PROCEDURES	ストリング	現行サーバーが SQL プロシージャーをサポートするかどうか。接続が SQL プロシージャーをサポートする場合は、値 ″Y″ が戻されます。
SQL_QUALIFIER_LOCATION または SQL_CATALOG_LOCATION	16 ビット整数	16 ビット整数値は、修飾表名の中の修飾子の位置を示します。ゼロは、修飾名がサポートされていないことを示します。
SQL_QUALIFIER_NAME_SEPARATOR または SQL_CATALOG_NAME_SEPARATOR	ストリング	カタログ名とその後に続く修飾名エレメントの間の区切り文字として使用される文字。

表 96. *SQLGetInfo* の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_QUALIFIER_TERM または SQL_CATALOG_TERM	ストリング	修飾子のデータベース・ベンダー用語 3 つの部分名のうちの高位部分にベンダーが用いる名前。 DB2 ODBC は 3 つの部分名をサポートしていないため、ゼロ長ストリングが戻されます。 非 ODBC アプリケーションの場合は、SQL_QUALIFIER_NAME の代わりに、シンボル名 SQL_CATALOG_TERM を使用する必要があります。
SQL_QUALIFIER_USAGE または SQL_CATALOG_USAGE	32 ビット・マスク	これは、カタログに使用されるという点以外は、SQL_OWNER_USAGE と類似しています。
SQL_QUOTED_IDENTIFIER_CASE	16 ビット整数	以下を戻します。 • SQL_IC_UPPER - SQL 内の引用符付き修飾子は大文字小文字を区別せず、システム・カタログに大文字で保管されます。 • SQL_IC_LOWER - SQL 内の引用符付き修飾子は大文字小文字を区別せず、システム・カタログに小文字で保管されます。 • SQL_IC_SENSITIVE - SQL 内の引用符付き修飾子 (区切り文字付き ID) は大文字小文字の区別があり、システム・カタログに大文字小文字混合で保管されます。 • SQL_IC_MIXED - SQL 内の引用符付き修飾子は大文字小文字を区別せず、システム・カタログに大文字小文字混合で保管されます。 これは、システム・カタログへの (引用符なしの) ID の保管方法を決定するために使用される SQL_IDENTIFIER_CASE fInfoType によって制約されます。
SQL_SEARCH_PATTERN_ESCAPE	ストリング	SQLTables()、SQLColumns() などのカタログ関数のエスケープ文字としてドライバーがサポートするものを指定するために使用されます。

表 96. *SQLGetInfo* の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_SQL92_PREDICATES	32 ビット・マスク	SQL-92 が定義する SELECT ステートメントでサポートされている述部を示します。 • SQL_SP_BETWEEN • SQL_SP_COMPARISON • SQL_SP_EXISTS • SQL_SP_IN • SQL_SP_ISNOTNULL • SQL_SP_ISNULL • SQL_SP_LIKE • SQL_SP_MATCH_FULL • SQL_SP_MATCH_PARTIAL • SQL_SP_MATCH_UNIQUE_FULL • SQL_SP_MATCH_UNIQUE_PARTIAL • SQL_SP_OVERLAPS • SQL_SP_QUANTIFIED_COMPARISON • SQL_SP_UNIQUE
SQL_SQL92_VALUE_EXPRESSIONS	32 ビット・マスク	SQL-92 が定義する、サポートされている値の式を示します。 • SQL_SVE_CASE • SQL_SVE_CAST • SQL_SVE_COALESCE • SQL_SVE_NULLIF

表 96. *SQLGetInfo* の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_STRING_FUNCTIONS	32 ビットのビット・マスク	サポートされているストリング関数を示します。 サポートされるストリング関数を決定するために、以下のビット・マスクが使用されます。 • 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 アプリケーションが、string1、string2、および開始引き数を伴う LOCATE スカラー関数を呼び出すことができる場合は、 SQL_FN_STR_LOCATE ビット・マスクが戻されます。アプリケーションが、string1 と string2 を伴う LOCATE スカラー関数のみを呼び出せる場合は、SQL_FN_STR_LOCATE_2 ビット・マスクが戻されます。 LOCATE スカラー関数が完全にサポートされている場合は、両方のビット・マスクが戻されます。

表 96. SQLGetInfo の戻り情報 (続き)

fInfoType	形式	説明と注
SQL_TIMEDATE_FUNCTIONS	32 ビット・マスク	サポートされている時間および日付関数を示します。 サポートされる日付関数を決定するために、以下のビット・マスクが使用されます。 • 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	トランザクションに DDL または DML (あるいはその両方) を含められるかどうかを示します。 • SQL_TC_NONE - トランザクションはサポートされていません。 • SQL_TC_DML - トランザクションには DML ステートメント (SELECT、INSERT、UPDATE、DELETE など) のみを含めることができます。トランザクション内に DDL ステートメント (CREATE TABLE、DROP INDEX など) が見つかると、エラーになります。 • SQL_TC_DDL_COMMIT - トランザクションには、DML ステートメントのみを含めることができます。トランザクション内に DDL ステートメントが見つかると、そのトランザクションはコミットされます。 • SQL_TC_DDL_IGNORE - トランザクションには、DML ステートメントのみを含めることができます。トランザクション内に DDL ステートメントが見つかっても、それは無視されます。 • SQL_TC_ALL - トランザクションには、 DDL ステートメントと DML ステートメントを任意の順序で含めることができます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 97. SQLGetInfo SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	要求された情報が NULL 文字終了ストリングとして戻され、長さが *cbInfoValueMax* に指定されているアプリケーション・プログラム・バッファーの長さを超えていました。引き数 *pcbInfoValue* の値は、要求された情報の実際の長さ (切り捨てられていない) になります。
08003	接続がオープンしていない	*fInfoType* で要求されているタイプの情報には、オープン接続が必要です。オープン接続が必要ないのは、SQL_ODBC_VER だけです。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *rgbInfoValue* は NULL ポインターです。 無効な *fInfoType* が指定されました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

SQLGetLength - ストリング値の長さの検索

目的

SQLGetLength() を使って、現在のトランザクション中にサーバーから戻された (取り出しまたは SQLGetSubString() 呼び出しの結果として) ラージ・オブジェクト・ロケーターが参照するラージ・オブジェクト値の長さを検索します。

構文

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

関数引き数

表 98. SQLGetLength の引き数

データ・タイプ	引き数	使用	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。準備作成されたステートメントをすでに割り振られているが、現在はそのステートメントを割り当てられていない任意のステートメント・ハンドルでかまいません。
SQLSMALLINT	*LocatorCType*	入力	C タイプのソース LOB ロケーター。次のいずれでもかまいません。 • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	入力	LOB ロケーター値に設定しなければなりません。
SQLINTEGER *	*StringLength*	出力	指定されたロケーターの長さ。[a] ポインターを NULL に設定すると、SQLSTATE **HY** 009 が戻されます。
SQLINTEGER *	*IndicatorValue*	出力	常にゼロに設定します。

注: a. DBCLOB データの場合でもバイト単位です。

使用法

SQLGetLength() を使うと、 LOB ロケーターが表すデータ値の長さを判別することができます。これをアプリケーション・プログラムで使って、参照されている LOB 値の全長を判別すれば、 LOB の一部または全部を取得するのに適した戦略をたてることができます。

Locator 引き数には任意の有効な LOB ロケーターを使うことができます。そのロケーターは、 FREE LOCATOR ステートメントで明示的に解放されたり、または、ロケーターを作成したトランザクションが終了したために暗黙で解放されたりしたものでなくてもかまいません。

このステートメント・ハンドルは、いずれかの準備作成されたステートメントや、カタログ関数呼び出しに関連付けられているものであってはなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 99. SQLGetLength SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効	*LocatorCType* と *Locator* の組み合わせは無効です。
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY003	プログラム・タイプが範囲外	*LocatorCType* は、 SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR、または SQL_C_DBCLOB_LOCATOR のいずれでもありません。
HY009	引き数値が無効	引き数 *StringLength* または *IndicatorValue* が NULL ポインターです。
HY010	関数シーケンス・エラー	指定した *StatementHandle* は、*割り振り済み* の状態にありません。
HYC00	ドライバーでサポートされていない	現在、アプリケーション・プログラムは、ラージ・オブジェクトをサポートしないデータ・ソースに接続されています。
0F001	LOB 変数が無効	*Locator* に指定した値は、LOB ロケーターに関連付けられていません。

制約事項

ラージ・オブジェクトをサポートしない DB2 サーバーに接続しているときは、この関数を使えません。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 108 ページの『SQLFetch - 次のデータ行』
- 176 ページの『SQLGetPosition - ストリングの開始位置を戻す』
- 184 ページの『SQLGetSubString - ストリング値の一部の検索』

SQLGetPosition - ストリングの開始位置を戻す

目的

SQLGetPosition() は、 LOB 値 (ソース) 内の 1 つのストリングの開始位置を戻すのに使います。ソース値は LOB ロケーターでなければなりませんが、検索ストリングは LOB ロケーターまたはリテラル・ストリングのどちらでもかまいません。

ソースおよび検索 LOB ロケーターは、現在のトランザクション中の取り出しまたは SQLGetSubString() 呼び出しでデータベースから戻された任意のロケーターでかまいません。

構文

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

関数引き数

表 100. SQLGetPosition の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。準備作成されたステートメントをすでに割り振られているが、現在はそのステートメントを割り当てられていない任意のステートメント・ハンドルでかまいません。
SQLSMALLINT	*LocatorCType*	入力	C タイプのソース LOB ロケーター。次のいずれでもかまいません。 • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*SourceLocator*	入力	*SourceLocator* は、ソース LOB ロケーターに設定しなければなりません。
SQLINTEGER	*SearchLocator*	入力	*SearchLiteral* ポインターが NULL の場合に、*SearchLiteralLength* を 0 に設定すると、*SearchLocator* を、検索ストリングに関連した LOB ロケーターに設定しなければなりません。そうしないと、この引き数は無視されます。
SQLCHAR *	*SearchLiteral*	入力	この引き数は、検索ストリング・リテラルを入れるストレージを指し示します。 *SearchLiteralLength* が 0 の場合、このポインターは NULL でなければなりません。
SQLINTEGER	*SearchLiteralLength*	入力	*SearchLiteral* 内のストリングの長さ (バイト数)。[a] この引き数値が 0 の場合、引き数 *SearchLocator* が妥当です。

表 100. SQLGetPosition の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*FromPosition*	入力	BLOB と CLOB の場合、これは、関数から戻される予定の、検索の開始地点であるソース・ストリング内の最初のバイトの位置になります。 DBCLOB の場合、これは先頭文字になります。先頭のバイトまたは文字には、番号 1 が付けられます。
SQLINTEGER *	*LocatedAt*	出力	BLOB と CLOB の場合、これは、ストリングが見つけ出されたバイト位置になります。ただし、見つからなかった場合、値はゼロになります。 DBCLOB の場合、これは文字位置になります。 ソース・ストリングの長さがゼロの場合、値 1 が戻されます。
SQLINTEGER *	*IndicatorValue*	出力	常にゼロに設定します。

注:

a DBCLOB データの場合でもバイト単位です。

使用法

SQLGetPosition() を SQLGetSubString() と一緒に使って、無作為にストリングの任意の部分を取得します。 SQLGetSubString() を使うには、ストリング全体の中のサブストリングの場所があらかじめ分かっている必要があります。検索ストリングを使って、サブストリングの開始地点を見つけられる場合、SQLGetPosition() を使えば、そのサブストリングの開始位置を取得することができます。

Locator および *SearchLocator* 引き数 (使用する場合) には、 FREE LOCATOR ステートメントで明示的に解放されたり、または、ロケーターを作成したトランザクションが終了したために暗黙で解放されたりしたものではないような、任意の有効な LOB ロケーターを使うことができます。

Locator と *SearchLocator* は、同じ LOB ロケーター・タイプでなければなりません。

このステートメント・ハンドルは、いずれかの準備作成されたステートメントや、カタログ関数呼び出しに関連付けられているものであってはなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 101. SQLGetPosition SQLSTATE

SQLSTATE	説明	解説
07006	変換が無効	*LocatorCType* と LOB ロケーター値の片方との組み合わせは無効です。

SQLGetPosition

表 101. SQLGetPosition SQLSTATE (続き)

SQLSTATE	説明	解説
42818	長さが無効	パターン長が長すぎます。
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY009	引き数値が無効	引き数 *LocatedAt* または *IndicatorValue* が NULL ポインターです。 *FromPosition* の引き数値が 0 より大きくありません。 *LocatorCType* は、 SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR、または SQL_C_DBCLOB_LOCATOR のどれでもありません。
HY010	関数シーケンス・エラー	指定した *StatementHandle* は、*割り振り済み* の状態にありません。
HY090	ストリングまたはバッファー長が無効	*SearchLiteralLength* の値は 1 より小さいですが、 SQL_NTS ではありません。
HYC00	ドライバーでサポートされていない	現在、アプリケーション・プログラムは、ラージ・オブジェクトをサポートしないデータ・ソースに接続されています。
0F001	LOB 変数が無効	*Locator* または *SearchLocator* に指定した値は、現在は LOB ロケーターではありません。

制約事項

ラージ・オブジェクトをサポートしない DB2 サーバーに接続しているときは、この関数を使えません。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 105 ページの『SQLExtendedFetch - 行配列の取り出し』
- 108 ページの『SQLFetch - 次のデータ行』
- 174 ページの『SQLGetLength - ストリング値の長さの検索』
- 184 ページの『SQLGetSubString - ストリング値の一部の検索』

SQLGetStmtAttr - ステートメント属性の値の取得

目的

SQLGetStmtAttr() は、指定されたステートメント属性の現行設定を戻します。

これらのオプションは、 SQLSetStmtAttr() 関数で設定されます。この関数は、 SQLGetStmtOption() オプションと類似していますが、どちらの関数も互換性の理由でサポートされています。

構文

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

関数引き数

表 102. SQLGetStmtAttr の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLINTEGER	*fAttr*	入力	検索する属性。詳細については、表 103 を参照してください。
SQLPOINTER	*pvParam*	出力	要求された属性のバッファーへのポインター。
SQLINTEGER	*bLen*	入力	属性が文字ストリングの場合、 *pvParam* に保管されるバイトの最大数。それ以外の場合は、使用されません。
SQLINTEGER *	*sLen*	出力	この属性が文字ストリングの場合、出力データの長さ。それ以外の場合は、使用されません。

使用法

表 103. ステートメント属性

fAttr	データ・タイプ	内容
SQL_ATTR_FOR_FETCH_ONLY	整数	このステートメント・ハンドルのオープン・カーソルは読み取り専用になっている必要があることを示します。 • SQL_FALSE - カーソルを、位置の決まった更新および削除に使用できる。これがデフォルトです。 • SQL_TRUE - カーソルは読み取り専用で、位置の決まった更新または削除には使用できない。
SQL_ATTR_APP_ROW_DESC	整数	ステートメント・ハンドルを使用して行データを検索するアプリケーション・プログラムの記述子ハンドル。

表 103. ステートメント属性 (続き)

fAttr	データ・タイプ	内容
SQL_ATTR_APP_PARAM_DESC	整数	このステートメント・ハンドルのパラメーター値を提供するときにアプリケーション・プログラムが使用する記述子ハンドル。
SQL_ATTR_CURSOR_SCROLLABLE	整数	このステートメント・ハンドルのカーソル・オープンをスクロール可能にするかどうかを指定する 32 ビット整数値。 • SQL_FALSE - カーソルをスクロール可能にしない。また、カーソルに対して SQLFetchScroll() を使用しない。これがデフォルトです。 • SQL_TRUE - カーソルをスクロール可能にする。これらのカーソルのデータ検索に、SQLFetchScroll() を使用できます。
SQL_ATTR_CURSOR_TYPE	整数	このステートメント・ハンドルに対して開かれるカーソルの動作を指定する 32 ビット整数値。 • SQL_CURSOR_FORWARD_ONLY - カーソルをスクロール可能にしない。また、カーソルに対して SQLFetchScroll() を使用しない。これがデフォルトです。 • SQL_DYNAMIC - カーソルをスクロール可能にする。これらのカーソルのデータ検索に、 SQLFetchScroll() を使用できます。
SQL_ATTR_IMP_ROW_DESC	整数	このステートメント・ハンドルを使用して行データを検索するときに CLI の実装で使用する記述子ハンドル。
SQL_ATTR_IMP_PARAM_DESC	整数	このステートメント・ハンドルのパラメーター値を提供するときに CLI の実装で使用する記述子ハンドル。
SQL_ATTR_ROWSET_SIZE	整数	行セット内の行数を指定する 32 ビット整数値。これは、 SQLExtendedFetch() の各呼び出しで戻される行数です。デフォルト値は 1 です。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 104. SQLStmtOption SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *pvParam* は NULL ポインターです。 指定された *fAttr* 値は無効です。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI はこのオプションを認識しますが、サポートはしていません。

SQLGetStmtOption - ステートメント・オプションの現行設定を戻す

目的

SQLGetStmtOption() は、指定されたステートメント・オプションの現行設定を戻します。

これらのオプションは、SQLSetStmtOption() 関数で設定されます。

構文

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

関数引き数

表 105. SQLStmtOption の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	接続ハンドル
SQLSMALLINT	*fOption*	入力	検索するオプション。詳細については、 179 ページの表 103 を参照してください。
SQLPOINTER	*pvParam*	出力	オプションの値。 *fOption* の値に応じ、32 ビット整数値、または NULL 終了文字ストリングへのポインターになります。

使用法

SQLGetStmtOption() は、 SQLGetStmtAttr() と同じ関数を提供していますが、どちらの関数も互換性の理由でサポートされています。

ステートメント・オプションの詳細については、 179 ページの表 103 を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 106. SQLStmtOption SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *pvParam* は NULL ポインターです。 指定された *fOption* 値は無効です。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI はこのオプションを認識しますが、サポートはしていません。

SQLGetSubString - ストリング値の一部の検索

目的

SQLGetSubString() を使って、現在のトランザクション中にサーバーから戻された (取り出しまたは直前の SQLGetSubString() 呼び出しで戻された) ラージ・オブジェクト・ロケーターが参照するラージ・オブジェクト値の一部を検索します。

構文

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

関数引き数

表 107. SQLGetSubString の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。準備作成されたステートメントをすでに割り振られているが、現在はそのステートメントを割り当てられていない任意のステートメント・ハンドルでかまいません。
SQLSMALLINT	*LocatorCType*	入力	C タイプのソース LOB ロケーター。次のいずれでもかまいません。 • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*SourceLocator*	入力	*SourceLocator* は、ソース LOB ロケーター値に設定しなければなりません。
SQLINTEGER	*FromPosition*	入力	BLOB と CLOB の場合、これは、関数から戻される予定の最初のバイトの位置になります。 DBCLOB の場合、これは先頭文字になります。 先頭のバイトまたは文字には、番号 1 が付けられます。
SQLINTEGER	*ForLength*	入力	これは、関数から戻される予定のストリングの長さです。 BLOB と CLOB の場合、これは、バイト単位の長さです。 DBCLOB の場合、これは文字数単位の長さです。 *FromPosition* が、ソース・ストリングの長さより短い場合に、 *FromPosition* + *ForLength* - 1 が、ソース・ストリングの終わりを越えると、その結果は、必要数の文字 (BLOB の場合は X'00'、CLOB の場合は単一バイトのブランク文字、 DBCLOB の場合は 2 バイトのブランク文字) と一緒に、右側に埋め込まれます。

表 107. SQLGetSubString の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*TargetCType*	入力	C データ・タイプの *DataPtr*。ターゲットは C ストリング変数 (SQL_C_CHAR、 SQL_C_WCHAR、SQL_C_BINARY、または SQL_C_DBCHAR) でなければなりません。
SQLPOINTER	*DataPtr*	出力	検索されたストリング値または LOB ロケーターを保管するバッファーへのポインター。
SQLINTEGER	*BufferLength*	入力	*DataPtr* が指すバッファーのバイト単位の最大サイズ。
SQLINTEGER *	*StringLength*	出力	ターゲットの C バッファー・タイプがバイナリーまたは文字ストリング変数用のものであって、ロケーター値ではない場合に、 *DataPtr* に戻されるバイト単位[a] の情報の長さ。 ポインターを NULL に設定すると、何も戻されません。
SQLINTEGER *	*IndicatorValue*	出力	常にゼロに設定します。

注:

a DBCLOB データの場合でもバイト単位です。

使用法

SQLGetSubString() は、 LOB ロケーターで表されるストリングの任意の部分を取得するのに使います。ターゲットには、次の 2 つの選択肢があります。

- 適切な C ストリング変数をターゲットにすることができます。
- 新規の LOB 値をサーバー上で作成し、その値の LOB ロケーターを、クライアント上のターゲット・アプリケーション・プログラム変数に割り当てることができます。

SQLGetSubString() を SQLGetData の代わりに使って、データを分割して入手することができます。その場合、列がまず LOB ロケーターにバインドされ、次にそれが使われて、その LOB の全部または一部が取り出されます。

Locator 引き数には任意の有効な LOB ロケーターを使うことができます。そのロケーターは、 FREE LOCATOR ステートメントで明示的に解放されたり、または、ロケーターを作成したトランザクションが終了したために暗黙で解放されたりしたものでなくてもかまいません。

このステートメント・ハンドルは、いずれかの準備作成されたステートメントや、カタログ関数呼び出しに関連付けられているものであってはなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 108. SQLGetSubString SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	戻そうとしているデータ量が *BufferLength* より長いです。戻すのに使用できる実際の長さは *StringLength* に保管されています。
07006	変換が無効	*TargetCType* に指定した値は、SQL_C_CHAR、 SQL_C_BINARY、SQL_C_DBCHAR または LOB ロケーターのいずれでもありません。 *TargetCType* に指定した値は、ソースには適していません (たとえば、BLOB 列に SQL_C_DBCHAR を指定しました)。
22011	サブストリングのエラーが発生	*FromPosition* は、ソース・ストリングの長さより大きいです。
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY003	プログラム・タイプが範囲外	*LocatorCType* は、 SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR、または SQL_C_DBCLOB_LOCATOR のどれでもありません。
HY009	引き数値が無効	*FromPosition* または *ForLength* に指定した値は、正整数ではありません。 引き数 *DataPtr*、*StringLength*、または *IndicatorValue* は NULL ポインターです。
HY010	関数シーケンス・エラー	指定した *StatementHandle* は、*割り振り済み* の状態にありません。
HY090	ストリングまたはバッファー長が無効	*BufferLength* の値は 0 未満です。
HYC00	ドライバーでサポートされていない	現在、アプリケーション・プログラムは、ラージ・オブジェクトをサポートしないデータ・ソースに接続されています。
0F001	現在ロケーターは未割り当て	*Locator* に指定した値は、現在は LOB ロケーターではありません。

制約事項

ラージ・オブジェクトをサポートしない DB2 サーバーに接続しているときは、この関数を使えません。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 108 ページの『SQLFetch - 次のデータ行』
- 143 ページの『SQLGetData - 列のデータの取得』
- 174 ページの『SQLGetLength - ストリング値の長さの検索』
- 176 ページの『SQLGetPosition - ストリングの開始位置を戻す』

SQLGetTypeInfo - データ・タイプ情報の入手

目的

SQLGetTypeInfo() は、 DB2 UDB CLI に関連した DBMS でサポートされているデータ・タイプに関する情報を戻します。その情報は、SQL 結果セットに戻されます。照会を処理するのに使うのと同じ関数を使って、列を受け取ることができます。

構文

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

関数引き数

表 109. SQLGetTypeInfo の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLSMALLINT	*DataType*	入力	照会対象の SQL データ・タイプ。サポートされているタイプは以下のとおりです。 • 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 SQL_ALL_TYPES を指定すると、サポートされているデータ・タイプに関するすべての情報が、TYPE_NAME 別の昇順で戻されます。結果セットには、サポートされていないどのデータ・タイプも入っていません。

使用法

SQLGetTypeInfo() は、結果セットを生成しますが、照会の実行と同じなので、カーソルを生成してトランザクションを開始します。このステートメント・ハンドル上で別のステートメントを準備作成して実行するには、このカーソルをクローズしなければなりません。

無効な *DataType* を指定して SQLGetTypeInfo() を呼び出すと、空の結果セットが戻されます。

以下に、この関数で生成される結果セットの列について説明します。

今後のリリースでは、新しい列が追加されたり、既存の列が変更されたりする可能性はありますが、現行列の位置は変更されません。戻されるデータ・タイプは、 CREATE TABLE、ALTER TABLE、DDL ステートメント内で使用できるものです。非持続データ・タイプは、戻される結果セット内には含まれません。ユーザー定義のデータ・タイプも戻されません。

表 110. SQLGetTypeInfo によって戻される列

列番号 / 列名	データ・タイプ	説明
1 TYPE_NAME	VARCHAR(128) NOT NULL	SQL データ・タイプ名の文字による表示 (例: VARCHAR、 DATE、 INTEGER)。
2 DATA_TYPE	SMALLINT NOT NULL	SQL データ・タイプ定義値 (例: SQL_VARCHAR、SQL_DATE、SQL_INTEGER)。
3 COLUMN_SIZE	INTEGER	データ・タイプが文字または 2 進ストリングの場合、この列には、バイト数の最大長が入ります。また、グラフィック (DBCS) ストリングの場合は、この列の 2 バイト文字数になります。 日付、時刻、タイム・スタンプのデータ・タイプの場合、これは、文字への変換後に値を表示するのに必要な合計文字数になります。 数値データ・タイプの場合、これは、合計桁数になります。
4 LITERAL_PREFIX	VARCHAR(128)	このデータ・タイプのリテラルの場合に、DB2 が接頭部と認識する文字。リテラルの接頭部が適用されない場合、この列のデータ・タイプは NULL になります。
5 LITERAL_SUFFIX	VARCHAR(128)	このデータ・タイプのリテラルの場合に、DB2 が接尾部と認識する文字。リテラルの接頭部が適用されない場合、この列のデータ・タイプは NULL になります。

表 110. SQLGetTypeInfo によって戻される列 (続き)

列番号 / 列名	データ・タイプ	説明
6 CREATE_PARAMS	VARCHAR(128)	この列のテキストには、コンマで区切られたキーワード・リストが入ります。それらのキーワードは、アプリケーション・プログラムが、 SQL におけるデータ・タイプとして TYPE_NAME 列内の名前を使うときに、小括弧で囲んで指定する各パラメーターに対応します。このリスト内のキーワードは、LENGTH、PRECISION、SCALE のいずれでもかまいません。キーワードは、 SQL 構文に定められている使用順序で並んでいます。 データ・タイプ定義 (INTEGER など) 用のパラメーターがない場合、 NULL 標識が戻されます。 **注:** CREATE_PARAMS の目的は、 *DDL ビルダー* のインターフェースをアプリケーション・プログラムにカスタマイズさせることにあります。アプリケーション・プログラムは、これを使うときは、データ・タイプを定義するのに必要な引き数の数を指定したり、編集制御にラベルを付けるのに使用できるローカル化テキストを備えたりすることしかできないことを承知しておく必要があります。
7 NULLABLE	SMALLINT NOT NULL	データ・タイプに NULL 値を使用できるかどうかを示します。 • NULL 値を禁止するには SQL_NO_NULLS に設定します。 • NULL 値を許可するには SQL_NULLABLE に設定します。
8 CASE_SENSITIVE	SMALLINT NOT NULL	データ・タイプを、照合目的で大文字小文字の区別があるものとして扱えるかどうかを示します。有効値は SQL_TRUE と SQL_FALSE です。
9 SEARCHABLE	SMALLINT NOT NULL	WHERE 文節内でのデータ・タイプの使用法を示します。有効値は次のとおりです。 • SQL_UNSEARCHABLE - WHERE 文節内でデータ・タイプを使用できません。 • SQL_LIKE_ONLY - LIKE 述部を付けた場合のみ、WHERE 文節内でデータ・タイプを使用できます。 • SQL_ALL_EXCEPT_LIKE - WHERE 文節内で、 LIKE 以外のすべての比較演算子と一緒にデータ・タイプを使用できます。 • SQL_SEARCHABLE - WHERE 文節内で、任意の比較演算子と一緒にデータ・タイプを使用できます。
10 UNSIGNED_ATTRIBUTE	SMALLINT	データ・タイプが符号なしかどうかを示します。有効値は SQL_TRUE、SQL_FALSE、または NULL です。この属性が適用されないデータ・タイプの場合、NULL 標識が戻されます。
11 FIXED_PREC_SCALE	SMALLINT NOT NULL	データ・タイプが、厳密な数値であって、常に同じ精度と位取りをもつ場合、値 SQL_TRUE が入ります。そうでなければ、SQL_FALSE が入ります。

表 110. SQLGetTypeInfo によって戻される列 (続き)

列番号 / 列名	データ・タイプ	説明
12 AUTO_INCREMENT	SMALLINT	行の挿入時に、このデータ・タイプの列が自動的に固有値に設定される場合は、SQL_TRUE が入ります。そうでなければ、SQL_FALSE が入ります。
13 LOCAL_TYPE_NAME	VARCHAR(128)	この列には、データ・タイプの通常名とは異なるローカル化された任意のデータ・タイプ名 (固有言語) が入ります。ローカル化された名前がない場合、この列は NULL になります。 この列は表示用でしかありません。ストリングの文字セットはロケールに準じますが、通常は、データベースのデフォルトの文字セットになります。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 111. SQLGetTypeInfo SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルは、ステートメント・ハンドル上ですでにオープンしています。 *StatementHandle* はまだクローズされていません。
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY004	SQL データ・タイプが範囲外	無効な *DataType* を指定しました。
HY010	関数シーケンス・エラー	data-at-execute (`SQLParamData()`、 `SQLPutData()`) の操作中に関数を呼び出しました。
HYT00	タイムアウト満了	

制約事項

次に示す ODBC 指定の SQL データ・タイプ (およびそれに対応する *DataType* 定義値) は、どの IBM RDBMS でもサポートされていません。

データ・タイプ	*DataType*
TINY INT	SQL_TINYINT
BIT	SQL_BIT

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 ) ;
```

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 159 ページの『SQLGetInfo - 一般情報の取得』

SQLLanguages - SQL 言語または準拠情報の取得

目的

SQLLanguages() は、SQL 言語または準拠情報を戻します。情報は SQL 結果セットに戻されますが、このセットは、SELECT ステートメントで生成された結果セットの取り出しに使用する関数と同じ関数で検索することができます。

構文

```
SQLRETURN SQLLanguages  (SQLHSTMT       hstmt);
```

関数引き数

表 112. SQLLanguages の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル

使用法

この関数は、ダイアレクトまたは準拠情報を StatementHandle の結果セットの形で戻します。ここには、当該の SQL 製品で明言されている準拠内容をすべて記述した行 (ISO 定義のサブセットおよびベンダー固有のバージョンなど) が入れられます。この仕様に準拠すると明言された製品の場合、結果セットの値は最低 1 行になります。

ISO 規格およびベンダー固有の言語を定義する行が同じ表に存在する可能性もあります。それぞれの行には、最低限これらの列が入っていますが、それが、X/Open SQL の準拠を明言する根拠となっている場合、これらの列の値は以下のようになります。

表 113. SQLLanguages によって戻される列

列名	データ・タイプ	説明
SOURCE	VARCHAR(254)、NOT NULL	この SQL バージョンを定義した組織。
SOURCE_YEAR	VARCHAR(254)	関連する原書類が承認された年。
CONFORMANCE	VARCHAR(254)	実装で明言されている関連書類への準拠レベル。
INTEGRITY	VARCHAR(254)	実装で Integrity Enhancement Feature (IEF) をサポートしているかどうかの指示。
IMPLEMENTATION	VARCHAR(254)	ベンダーの SQL 製品を固有に識別するための、ベンダー定義の文字ストリング。
BINDING_SYTLE	VARCHAR(254)	'EMBEDDED'、'DIRECT'、または 'CLI' のいずれか。
PROGRAMMING_LANG	VARCHAR(254)	バインド形式がサポートされているホスト言語。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 114. SQLLanguages SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルに関する情報を要求しましたが、オープンされたカーソルはありません。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI では、表名の修飾子として *catalog* をサポートしていません。

SQLMoreResults - さらに結果セットがあるかどうかの判別

目的

SQLMoreResults() は、結果セットを戻すストアード・プロシージャーに関連付けられているステートメント・ハンドル上に、入手可能な情報がさらにあるかどうかを判別します。

構文

```
SQLRETURN   SQLMoreResults   (SQLHSTMT          StatementHandle);
```

関数引き数

表 115. SQLMoreResults の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。

使用法

この関数を使って、 SQL 照会の入っているストアード・プロシージャーの実行時に、順次設定されている複数の結果セットを戻します。ストアード・プロシージャーの実行が完了しても、結果セットをアクセス可能のままにしておくため、カーソルはオープンしたままになります。

アプリケーション・プログラムは、最初の結果セットの処理後、 SQLMoreResults() を呼び出して、別の結果セットを入手できるかどうかを判別することができます。現在の結果セット内にまだ取り出していない行がある場合、 SQLMoreResults() は、カーソルをクローズしてそのような行を破棄してから、さらに別の結果セットがあれば、SQL_SUCCESS を戻します。

すべての結果セットの処理が終わったら SQLMoreResults() は、 SQL_NO_DATA_FOUND を戻します。

SQL_CLOSE または SQL_DROP オプションを指定して SQLFreeStmt() を呼び出すと、このステートメント・ハンドル上の保留中の結果セットはすべて廃棄されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

エラー状況

表 116. SQLMoreResults SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。

表 116. SQLMoreResults SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	data-at-execute (SQLParamData()、 SQLPutData()) の操作中に関数を呼び出しました。
HY013	予想外のメモリー処理エラー	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーにアクセスできません。
HYT00	タイムアウト満了	

さらに、SQLMoreResults() は、 SQLExecute() に関連した SQLSTATE を戻すこともできます。

制約事項

また、SQLMoreResults() の ODBC 仕様を使って、入力パラメーター値の配列をもつパラメーター化された INSERT、UPDATE、および DELETE ステートメントの実行に関連したカウント値を戻すこともできます。ただし、DB2 UDB CLI では、このようなカウント情報の戻りはサポートされません。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 53 ページの『SQLBindParameter - バッファーに対するパラメーター・マーカーのバインド』

SQLNativeSql - 固有の SQL テキストの入手

目的

SQLNativeSql() は、ベンダーのエスケープ文節の DB2 UDB CLI での解釈法を示すのに使います。アプリケーション・プログラムが渡した元の SQL ストリングに、ベンダーのエスケープ文節が入っていた場合、DB2 UDB CLI は、データ・ソースで見られるとおりの変換後の SQL ストリング (適宜、ベンダーのエスケープ文節を変換または廃棄してから) を戻します。

構文

```
SQLRETURN   SQLNativeSql     (SQLHDBC          ConnectionHandle,
                              SQLCHAR          *InStatementText,
                              SQLINTEGER       TextLength1,
                              SQLCHAR          *OutStatementText,
                              SQLINTEGER       BufferLength,
                              SQLINTEGER       *TextLength2Ptr);
```

関数引き数

表 117. SQLNativeSql の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	ConnectionHandle	入力	接続ハンドル
SQLCHAR *	InStatementText	入力	入力 SQL ストリング。
SQLINTEGER	TextLength1	入力	*InStatementText* の長さ。
SQLCHAR *	OutStatementText	出力	変換後の出力ストリング用のバッファーを指すポインター。
SQLINTEGER	BufferLength	入力	*OutStatementText* が指すバッファーのサイズ。
SQLINTEGER *	TextLength2Ptr	出力	*OutStatementText* に戻せる合計バイト数。戻すのに使用できるバイト数が *BufferLength* より大か等しい場合、 *OutStatementText* 内の出力 SQL ストリングは、 *BufferLength - 1* バイトに切り捨てられます。出力ストリングが生成されない場合は、値 SQL_NULL_DATA が戻されます。

使用法

この関数を呼び出すのは、 DB2 UDB CLI からデータ・ソースに渡される変換後の SQL ストリングを、アプリケーション・プログラムで検査または表示したい場合です。変換 (マッピング) が行われるのは、ベンダーのエスケープ文節シーケンスが入力 SQL ステートメントに入っている場合だけです。

iSeries ではベンダー・エスケープ・シーケンスはありません。ここに示す手順は、互換性のためでしかありません。また、この手順を使って、 SQL ストリングの構文エラーを見つけることもできます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 118. SQLNativeSql SQLSTATE

SQLSTATE	説明	解説
01004	データは切り捨てられる	*OutStatementText* は、 SQL ストリング全体を入れるのに十分な大きさではないので、切り捨てが行われました。引き数 *TextLength2Ptr* には、切り捨てられていない SQL ストリングの全長が入ります。 (この関数は SQL_SUCCESS_WITH_INFO を戻します。)
08003	接続はクローズ済み	*ConnectionHandle* は、オープンしているデータベース接続を参照していません。
37000	SQL 構文が無効	*InStatementText* 内の入力 SQL ストリングには、構文エラーがあります。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *InStatementText*、*OutStatementText*、または *TextLength2Ptr* は NULL ポインターです。
HY090	ストリングまたはバッファー長が無効	引き数 *TextLength1* は 0 未満ですが、 SQL_NTS に等しくありません。 引き数 *BufferLength* は 0 未満です。

制約事項

なし。

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 ) ;
```

参照

なし。

SQLNextResult - 次の結果セットの処理

目的

SQLNextResult() は、結果セットを戻すストアード・プロシージャーに関連付けられているステートメント・ハンドル上に、入手可能な情報がさらにあるかどうかを判別します。

構文

```
SQLRETURN   SQLNextResult   (SQLHSTMT          StatementHandle,
                             SQLHSTMT          NextResultHandle);
```

関数引き数

表 119. SQLNextResult の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLHSTMT	NextResultHandle	入力	次の結果セットのステートメント・ハンドル。

使用法

この関数は、StatementHandle からの次の結果セットを NextResultHandle に関連付けるために使用されます。 SQLMoreResults() と異なり、両方のステートメント・ハンドルがそれらの結果セットを同時に処理できます。

すべての結果セットの処理が終わったら SQLNextResult() は、 SQL_NO_DATA_FOUND を戻します。

SQL_CLOSE または SQL_DROP オプションを指定して SQLFreeStmt() を呼び出すと、このステートメント・ハンドル上の保留中の結果セットはすべて廃棄されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

エラー状況

表 120. SQLNextResult SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
58004	想定外のシステム障害	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	data-at-execute (SQLParamData()、SQLPutData()) の操作中に関数を呼び出しました。

表 120. SQLNextResult SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予想外のメモリー処理エラー	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーにアクセスできません。
HYT00	タイムアウト満了	

参照

- 194 ページの『SQLMoreResults - さらに結果セットがあるかどうかの判別』

SQLNumParams - SQL ステートメント内のパラメーター数の入手

目的

SQLNumParams() は、 SQL ステートメント内のパラメーター・マーカー数を戻します。

構文

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       *ParameterCountPtr);
```

関数引き数

表 121. SQLNumParams の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLSMALLINT *	ParameterCountPtr	出力	ステートメント内のパラメーターの数。

使用法

StatementHandle に関連したステートメントの準備後にのみ、この関数を呼び出すことができます。パラメーター・マーカーがステートメント内に入っていない場合、 *ParameterCountPtr* を 0 に設定します。

アプリケーション・プログラムは、この関数を呼び出して、ステートメント・ハンドルに関連した SQL ステートメントに、何回の SQLBindParameter() 呼び出しが必要かを判別することができます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 122. SQLNumParams SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作取り消し	
HY009	引き数値が無効	*ParameterCountPtr* は NULL です。
HY010	関数シーケンス・エラー	指定された *StatementHandle* に対して SQLPrepare() を呼び出す前に、この関数を呼び出しました。 data-at-execute (SQLParamData()、SQLPutData()) の操作中に関数を呼び出しました。

SQLNumParams

表 122. SQLNumParams SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予想外のメモリー処理エラー	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーにアクセスできません。
HYT00	タイムアウト満了	

制約事項

なし。

例

197 ページの『例』 (SQLNativeSql()) を参照してください。

参照

- 48 ページの『SQLBindParam - パラメーター・マーカーに対するバッファーのバインド』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLNumResultCols - 結果列の数の取得

目的

SQLNumResultCols() は、入力ステートメント・ハンドルと関連する結果セットに列数を戻します。

この関数の前に、 SQLPrepare() または SQLExecDirect() を呼び出す必要があります。

この関数を呼び出した後で、SQLDescribeCol()、SQLColAttributes()、SQLBindCol()、または SQLGetData() を呼び出すことができます。

構文

```
SQLRETURN SQLNumResultCols (SQLHSTMT       hstmt,
                            SQLSMALLINT    *pccol);
```

関数引き数

表 123. SQLNumResultCols の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT *	*pccol*	出力	結果セットの列の数

使用法

入力ステートメント・ハンドルで実行された最後のステートメントが SELECT でない場合、この関数は出力引き数をゼロに設定します。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 124. SQLNumResultCols SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	*pcbCol* が NULL ポインターになっていました。
HY010	関数シーケンス・エラー	*hstmt* に対し、 SQLPrepare または SQLExecDirect より先にこの関数が呼び出されました。
S1013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

SQLNumResultCols

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 63 ページの『SQLColAttributes - 列属性』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 130 ページの『SQLGetCol - 結果セットの行での 1 つの列の検索』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLParamData - データ値が必要な次のパラメーターの取得

目的

SQLParamData() は、 SQLPutData( ) と組み合わせて、長いデータを断片的に送信する場合に使用します。また、固定長データの送信にも使用できます。

構文

```
SQLRETURN SQLParamData  (SQLHSTMT     hstmt,
                         SQLPOINTER   *prgbValue);
```

関数引き数

表 125. SQLParamData の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLPOINTER *	*prgbValue*	出力	SQLSetParam 呼び出しに指定されている *rgbValue* 引き数値へのポインター。

使用法

データが割り当てられていない SQL_DATA_AT_EXEC パラメーターが 1 つでもあると、SQLParamData() は SQL_NEED_DATA を戻します。この関数は、直前の SQLBindParam() 呼び出し時に、アプリケーション・プログラムから提供される *prgbValue* にアプリケーション・プログラム定義の値を戻します。 SQLPutData() を何回か呼び出して、パラメーター・データを送信します。 SQLParamData() は、現行パラメーターのすべてのデータが送信されると信号を出し、次の SQL_DATA_AT_EXEC パラメーターに進みます。すべてのパラメーターにデータ値が割り当てられ、関連ステートメントが正常実行されると、SQL_SUCCESS が戻されます。実際のステートメント実行のときまたはその前にエラーが発生すると、SQL_ERROR が戻されます。

SQLParamData() が SQL_NEED_DATA を戻す場合に呼び出せるのは、 SQLPutData() または SQLCancel() だけです。このステートメント・ハンドルを使用して呼び出す他の関数は、すべて失敗します。さらに、*hstmt* の親 *hdbc* を参照する関数呼び出しも、その接続の属性または状態の変更に関係している場合は、すべて失敗します。親 *hdbc* に対する以下の関数呼び出しも許可されていません。

- SQLAllocConnect()
- SQLAllocHandle()
- SQLAllocStmt()
- SQLSetConnectOption()

これらの関数が SQL_NEED_DATA 順序列で呼び出されると、これらの関数は SQLSTATE が **HY**010 の SQL_ERROR を戻しますが、 SQL_DATA_AT_EXEC パラメーターの処理に影響はありません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

- SQL_NEED_DATA

診断

SQLParamData() の戻り値としては、 SQLExecDirect() および SQLExecute() 関数が戻す SQLSTATE ならばすべて有効です。さらに、以下の診断も生成できます。

表 126. SQLParamData SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *prgbValue* は NULL ポインターです。
HY010	関数シーケンス・エラー	SQLParamData() が順序外で呼び出されました。この呼び出しが有効なのは、 SQLExecDirect() か SQLExecute() の後、または SQLPutData() 呼び出しの後に呼び出す場合に限られます。
HYDE0	実行が保留されているデータ値がない	この関数は、 SQLExecDirect() または SQLExecute() 呼び出しの後に呼び出されましたが、処理する SQL_DATA_AT_EXEC パラメーターがありませんでした。

SQLParamOptions - パラメーターの入力配列の指定

目的

SQLParamOptions() には、 SQLBindParameter() で設定されたパラメーターごとに複数の値を設定する機能が備わっています。これを使ってアプリケーション・プログラムは、 SQLExecute() または SQLExecDirect() の 1 回の呼び出しで、複数の行を表に挿入できます。

構文

```
SQLRETURN   SQLParamOptions  (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Crow,
                              SQLINTEGER        *FetchOffsetPtr);
```

関数引き数

表 127. SQLParamOptions の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLINTEGER	*Crow*	入力	各パラメーターの値の数。これが 1 より大きい場合、 SQLBindParameter() の *rgbValue* 引き数はパラメーター値の配列を指し、 *pcbValue* は長さの配列を指します。
SQLINTEGER *	*FetchOffsetPtr*	出力 (据え置き)	現在は使用されていません。

使用法

この関数を SQLBindParameter() と一緒に使って、複数行の INSERT ステートメントをセットアップすることができます。そのためには、アプリケーション・プログラムは、挿入しようとするすべてのデータにストレージを割り振る必要があります。そのデータは、行に準じた方式で編成されていなければなりません。つまり、1 行目のデータはすべて連続していて、その後に次の行のすべてのデータが続き、その後同じように続くということです。すべての入力パラメーターのタイプと長さをバインドするには、SQLBindParameter() 関数を使用しなければなりません。複数行の INSERT ステートメントの場合、SQLBindParameter() に指定したアドレスが、第 1 行目のデータを参照するのに使われます。その後に続くどのデータ行も、その行全体の長さが加えられて順に増大するアドレスで参照されます。

たとえばアプリケーション・プログラムが、 100 行のデータを表に挿入する予定の場合に、各行に 4 バイトの整数値が入っていて、その後に 10 バイト文字値が続いているとします。アプリケーション・プログラムは、1400 バイトのストレージを割り振ってから、 14 バイトの各ストレージ部分に、行ごとの該当データを入れることになります。

また、SQLBindParameter() で渡す標識ポインターは、 800 バイトのストレージ部分も参照する必要があります。これは、すべての NULL 標識値を渡すのに使います。このストレージも行に準じているので、最初の 8 バイトは、 1 行目の 2 つの標識になり、その後に次の行の 2 つの標識が続き、以後同じように続きます。アプリケーション・プログラムは、SQLParamOptions() 関数を使って、ステートメント・ハンドルを使った次の INSERT ステートメントの実行時に何行を挿入するかを指定します。 INSERT ステートメントは、複数行形式でなければなりません。

SQLParamOptions

以下に例を示します。
INSERT INTO CORPDATA.NAMES ? ROWS VALUES(?, ?)

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 128. SQLParamOptions SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *Crow* 内の値が 1 より小さいです。
HY010	関数シーケンス・エラー	data-at-execute (`SQLParamData()`、`SQLPutData()`) の操作中に関数を呼び出しました。

制約事項

なし。

参照

- 48 ページの『SQLBindParam - パラメーター・マーカーに対するバッファーのバインド』
- 194 ページの『SQLMoreResults - さらに結果セットがあるかどうかの判別』

SQLPrepare - ステートメントの準備作成

目的

SQLPrepare() は、 SQL ステートメントを入力ステートメント・ハンドルと関連付け、このステートメントを DBMS に送信して準備作成します。アプリケーション・プログラムは、他の関数にステートメント・ハンドルを渡すことで、この準備作成されたステートメントを参照することができます。

ステートメント・ハンドルが SELECT ステートメントを指定して使用されている場合は、 SQLPrepare() より先に SQLFreeStmt() を呼び出して、カーソルをクローズする必要があります。

構文

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

関数引き数

表 129. SQLPrepare の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル。 *hstmt* に関連するオープン・カーソルは無効です。
SQLCHAR *	*szSqlStr*	入力	SQL ステートメント・ストリング。
SQLINTEGER	*cbSqlStr*	入力	*szSqlStr* 引き数の内容の長さ。 この長さは、 *szSqlstr* の SQL ステートメントの正確な長さに設定する必要がありますが、ステートメント・テキストが NULL 文字で終了している場合は SQL_NTS に設定する必要があります。

使用法

アプリケーション・プログラムは、 SQLPrepare() でステートメントを準備作成し終わったら、次のような関数を呼び出して、結果セットの形式 (SELECT ステートメントの場合) に関する情報を要求することができます。

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttributes()

準備作成されたステートメントは、1 回実行しても、または、 SQLExecute() を呼び出して複数回実行してもかまいません。この SQL ステートメントは、ステートメント・ハンドルが再び SQLPrepare()、SQLExecDirect()、SQLColumns()、SQLSpecialColumns()、SQLStatistics()、または SQLTables() で使用されるまで、このハンドルに関連付けられたままになります。

SQL ステートメントの値としては、パラメーター・マーカーも有効です。パラメーター・マーカーは、SQL ステートメントでは ″?″ 文字で表示され、 SQLExecute() の呼び出し時にアプリケーション・プログラム変数値に置換するステートメント内の桁位置を表します。 SQLBindParam() は、アプリケーション・プ

SQLPrepare

ログラム変数をそれぞれのパラメーター・マーカーにバインド (または関連付け) し、データ転送時に実行する必要のあるデータ変換があるかどうかを示します。

SQL ステートメントは、COMMIT または ROLLBACK できません。 COMMIT または ROLLBACK を発行するには、SQLTransact() を呼び出してください。

SQL ステートメントが位置の決まった DELETE または位置の決まった UPDATE である場合、このステートメントが参照するカーソルは、同じ接続ハンドルで別のステートメント・ハンドルに定義される必要があります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 130. SQLPrepare SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	指定された *hstmt* にオープン・カーソルがありました。
37xxx	構文エラーまたはアクセス違反	*szSqlStr* が、以下の 1 つ以上の値になっています。 • COMMIT • ROLLBACK • 接続されているデータベース・サーバーでは準備作成できない SQL ステートメント • 構文エラーのあるステートメント
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	*szSqlStr* が NULL ポインターになっていました。 引き数 *cbSqlStr* は 1 未満でしたが、 SQL_NTS と同じになっていませんでした。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

注: すべての DBMS が、準備作成時に上記のすべての診断メッセージを報告するわけではありません。このため、アプリケーション・プログラムは、 SQLExecute() を呼び出す場合は、これらの条件も処理できるようにしておく必要があります。

例

以下の例で使用されている check_error、initialize、および terminate 関数のリストについては、 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』を参照してください。

```
/*****************************************************************************
** 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>
#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 )
```

SQLPrepare

```
            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 */
```

参照

- 63 ページの『SQLColAttributes - 列属性』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 203 ページの『SQLNumResultCols - 結果列の数の取得』

SQLPrimaryKeys - 表の基本キー列の入手

目的

SQLPrimaryKeys() は、表の基本キーを構成する列名のリストを戻します。情報は SQL 結果セットに戻されますが、これは、照会で生成された結果セットの処理に使用するのと同じ関数を使って検索することができます。

構文

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

関数引き数

表 131. SQLPrimaryKeys の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLCHAR *	CatalogName	入力	3 分割の表名のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。
SQLCHAR *	SchemaName	入力	表名のスキーマ修飾子。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	TableName	入力	表名。
SQLSMALLINT	NameLength3	入力	*TableName* の長さ。

使用法

SQLPrimaryKeys() は、 1 つの表の基本キー列を戻します。スキーマ修飾子や表名を指定するのに、検索パターンを使うことはできません。

結果セットには、 表 132 に示されている列が入っています。その順序は、 TABLE_CAT、 TABLE_SCHEM、 TABLE_NAME、および ORDINAL_POSITION です。

多くの場合、SQLPrimaryKeys() の呼び出しは、システム・カタログに対する複雑な (そのため、経費のかさむ) 照会にマップされるので、慎重に使用する必要があり、何回も呼び出さなくて済むように結果を保管しておかなければなりません。

今後のリリースでは、新しい列が追加されたり、既存の列が変更されたりする可能性はありますが、現行列の位置は変更されません。

表 132. SQLPrimaryKeys によって戻される列

列番号 / 列名	データ・タイプ	説明
1 TABLE_CAT	VARCHAR(128)	現行サーバー。

表 132. SQLPrimaryKeys によって戻される列 (続き)

列番号 / 列名	データ・タイプ	説明
2 TABLE_SCHEM	VARCHAR(128)	TABLE_NAME が入っているスキーマの名前。
3 TABLE_NAME	NULL 以外の VARCHAR(128)	指定した表の名前。
4 COLUMN_NAME	NULL 以外の VARCHAR(128)	基本キーの列名。
5 ORDINAL_POSITION	NULL 以外の SMALLINT	基本キー内の、1 から始まる列順序番号。
6 PK_NAME	VARCHAR(128)	基本キー ID。データ・ソースに対して該当しない場合は NULL。

注: DB2 UDB CLI で使われる列名は、X/Open CLI CAE 仕様スタイルに準拠します。列のタイプ、内容、および順序は、 ODBC において `SQLPrimaryKeys()` の結果セット用に定義されているものと同じです。

指定した表に基本キーが入っていないと、空の結果セットが戻されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 133. SQLPrimaryKeys SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルは、ステートメント・ハンドル上ですでにオープンしています。
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作取り消し	
HY010	関数シーケンス・エラー	data-at-execute (`SQLParamData()`、`SQLPutData()`) の操作中に関数を呼び出しました。
HY014	ハンドルが不足	内部リソースに起因して DB2 UDB CLI はハンドルを割り振れませんでした。
HY090	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、 SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI では、表名の修飾子として *catalog* をサポートしていません。
HYT00	タイムアウト満了	

制約事項

なし。

参照

- 116 ページの『SQLForeignKeys - 外部キー列リストの入手』
- 262 ページの『SQLStatistics - 基本表の索引情報と統計情報の取得』

SQLProcedureColumns - プロシージャーの入出力パラメーター情報の入手

目的

SQLProcedureColumns() は、プロシージャーに関連した入出力パラメーターのリストを戻します。情報は SQL 結果セットに戻されますが、これは、照会で生成された結果セットの処理に使用するのと同じ関数を使って検索することができます。

構文

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

関数引き数

表 134. SQLProcedureColumns の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLCHAR *	CatalogName	入力	3 分割のプロシージャー名のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。 0 に設定してください。
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾する *pattern-value* を保管する可能性のあるバッファー。 DB2 UDB for zOS および OS/390 V 4.1 の場合、すべてのストアード・プロシージャーは 1 つのスキーマになっています。 *SchemaName* 引き数に使用できる唯一の値は NULL ポインターです。 DB2 Universal Database™ の場合、 *SchemaName* には有効なパターン値を入れることができます。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	ProcName	入力	プロシージャー名で結果セットを修飾する *pattern-value* を保管できるバッファー。
SQLSMALLINT	NameLength3	入力	*ProcName* の長さ。
SQLCHAR *	ColumnName	入力	パラメーター名で結果セットを修飾する *pattern-value* を保管できるバッファー。この引き数は、空でない値を ProcName または SchemaName に指定することで、すでに制限を受けている結果セットをさらに修飾するのに使います。
SQLSMALLINT	NameLength4	入力	*ColumnName* の長さ。

使用法

DB2 UDB CLI は、ストアード・プロシージャーに関連した入力、入出力、および出力パラメーターに関する情報を戻しますが、戻された結果セットの記述子情報に関する情報を戻すことはできません。

SQLProcedureColumns() は、 PROCEDURE_CAT、 PROCEDURE_SCHEM、 PROCEDURE_NAME、および COLUMN_TYPE の順で結果セット内の情報を戻します。結果セットの列は、表 135 にリストされています。アプリケーション・プログラムでは、今後のリリースで、最終列の後に列が定義される可能性のあることに注意する必要があります。

多くの場合、SQLProcedureColumns() の呼び出しは、システム・カタログに対する複雑な (そのため、経費のかさむ) 照会にマップされるので、慎重に使用する必要があり、何回も呼び出さなくて済むように結果を保管しておかなければなりません。

表 135. SQLProcedureColumns から戻される列

列番号 / 列名	データ・タイプ	説明
1 PROCEDURE_CAT	VARCHAR(128)	現行サーバー。
2 PROCEDURE_SCHEM	VARCHAR(128)	PROCEDURE_NAME が入っているスキーマの名前。
3 PROCEDURE_NAME	VARCHAR(128)	プロシージャーの名前。
4 COLUMN_NAME	VARCHAR(128)	パラメーターの名前。
5 COLUMN_TYPE	NULL 以外の SMALLINT	この行に関連したタイプ情報を識別します。値は次のいずれかになります。 • SQL_PARAM_TYPE_UNKNOWN - パラメーター・タイプは不明です。 **注:** これは戻されません。 • SQL_PARAM_INPUT - このパラメーターは入力パラメーターです。 • SQL_PARAM_INPUT_OUTPUT - このパラメーターは入出力パラメーターです。 • SQL_PARAM_OUTPUT - このパラメーターは出力パラメーターです。 • SQL_RETURN_VALUE - プロシージャー列は、そのプロシージャーの戻り値です。 **注:** これは戻されません。 • SQL_RESULT_COL - このパラメーターは、実際には結果セット内の列です。 **注:** これは戻されません。
6 DATA_TYPE	NULL 以外の SMALLINT	SQL データ・タイプ。
7 TYPE_NAME	NULL 以外の VARCHAR(128)	DATA_TYPE に対応するデータ・タイプの名前を表す文字ストリング。

表 135. SQLProcedureColumns から戻される列 (続き)

列番号 / 列名	データ・タイプ	説明
8 COLUMN_SIZE	INTEGER	DATA_TYPE 列値が文字または 2 進ストリングを表す場合、この列には、バイト数の最大長が入ります。また、グラフィック (DBCS) ストリングの場合は、パラメーターの 2 バイト文字数になります。 日付、時刻、タイム・スタンプのデータ・タイプの場合、これは、文字への変換後に値を表示するのに必要な合計バイト数になります。 数値データ・タイプの場合、これは、結果セット内の NUM_PREC_RADIX 列の値に応じて、合計桁数になるか、またはその列に使用できる合計ビット数になります。
9 BUFFER_LENGTH	INTEGER	SQLBindCol()、SQLGetData()、および SQLBindParameter() の呼び出し時に SQL_C_DEFAULT が指定された場合に、関連した C バッファーがこのパラメーターからデータを保管するバイトの最大数。その長さには、NULL 終了文字は含まれません。厳密な数値データ・タイプの場合、長さには小数部と符号も含まれます。
10 DECIMAL_DIGITS	SMALLINT	パラメーターの位取り。位取りが該当しないデータ・タイプの場合は、 NULL が戻されます。
11 NUM_PREC_RADIX	SMALLINT	10 または 2 または NULL のいずれか。 DATA TYPE が推定の数値データ・タイプである場合、この列には 2 が入れられ、COLUMN_SIZE 列には、このパラメーターで許可されているビット数が入れられます。 DATA_TYPE が厳密なデータ・タイプである場合、この列には値 10 が入れられ、 COLUMN_SIZE と DECIMAL_DIGITS の各列には、このパラメーターで許可されている 10 進数字の数が入れられます。 数値データ・タイプの場合、 DBMS から 10 または 2 の NUM_PREC_RADIX が戻されることがあります。 基数が該当しないデータ・タイプの場合は、NULL が戻されます。
12 NULLABLE	VARCHAR(3)	このパラメーターで NULL 値が受け入れられない場合は、'NO'。 このパラメーターで NULL 値が受け入れられる場合は、'YES'。
13 REMARKS	VARCHAR(254)	このパラメーターに関する記述情報が入れられる場合があります。

表 135. SQLProcedureColumns から戻される列 (続き)

列番号 / 列名	データ・タイプ	説明
14 COLUMN_DEF	VARCHAR	列のデフォルト値。 デフォルト値として NULL が指定された場合、この列は引用符なしのワード NULL になります。デフォルト値を切り捨てなければ表示できない場合、この列の値は単一引用符なしの TRUNCATED になります。デフォルト値が指定されない場合、この列の値は NULL になります。 COLUMN_DEF の値は、新しい列の定義を生成するために使用できます。ただし、値が TRUNCATED の場合は除きます。
15 SQL_DATA_TYPE	NULL 以外の SMALLINT	記述子の SQL_DESC_TYPE フィールドに現れるとおりの、SQL データ・タイプの値。この列は、日時データ・タイプを除き、DATA_TYPE 列と同じです (DB2 UDB CLI は時間間隔データ・タイプをサポートしていません)。 日時データ・タイプの場合、結果セットの SQL_DATA_TYPE フィールドは SQL_DATETIME になり、SQL_DATETIME_SUB フィールドは特定の日時データ・タイプのサブコードを戻します (SQL_CODE_DATE、SQL_CODE_TIME、または SQL_CODE_TIMESTAMP)。
16 SQL_DATETIME_SUB	SMALLINT	日時データ・タイプのサブタイプ・コード。他のすべてのデータ・タイプの場合、この列は NULL を戻します。(時間間隔データ・タイプを含みます。DB2 UDB CLI はこれをサポートしていません。)
17 CHAR_OCTET_LENGTH	INTEGER	文字データ・タイプ列の最大長 (バイト単位)。他のすべてのデータ・タイプの場合、この列は NULL を戻します。
18 ORDINAL_POSITION	NULL 以外の INTEGER	COLUMN_NAME で指定されたパラメーターが、この結果セットの中で占める位置を表す順番。これは、CALL ステートメントに引き数が指定される順番を示します。左端の引き数の順番が 1 になります。
19 IS_NULLABLE	VARCHAR	• 列に NULL が含まれない場合は “NO”。 • 列に NULL を含めることができる場合は “YES”。 • NULL 可能かどうかが不明の場合は、ゼロ長のストリング。 NULL 可能性の判別は、ISO の規則に従います。 ISO SQL 準拠の DBMS は、空ストリングを戻すことができません。 この列に戻される値は、NULLABLE 列に戻される値とは異なります。 (NULLABLE 列の説明を参照してください。)

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 136. SQLProcedureColumns SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルは、ステートメント・ハンドル上ですでにオープンしています。
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
42601	PARMLIST 構文エラー	ストアード・プロシージャーのカタログ表内の PARMLIST 値に、構文エラーがあります。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作取り消し	
HY010	関数シーケンス・エラー	
HY014	ハンドルが不足	内部リソースに起因して DB2 UDB CLI はハンドルを割り振れませんでした。
HY090	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、 SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI では、プロシージャー名の修飾子として *catalog* をサポートしていません。 接続先のサーバーは、プロシージャー名の修飾子として *schema* をサポートしていません。
HYT00	タイムアウト満了	

制約事項

SQLProcedureColumns() は、ストアード・プロシージャーから戻される可能性のある結果セットの属性に関する情報を戻しません。

アプリケーション・プログラムが、ストアード・プロシージャーのカタログをサポートしない DB2 に接続されているか、またはストアード・プロシージャーをサポートしない場合は、 SQLProcedureColumns() は空の結果セットを戻します。

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 */
```

参照

- 222 ページの『SQLProcedures - プロシージャー名リストの入手』

SQLProcedures - プロシージャー名リストの入手

目的

SQLProcedures() は、サーバーに登録されていて、しかも指定の検索パターンに一致するプロシージャー名のリストを戻します。

情報は SQL 結果セットに戻されますが、これは、照会で生成された結果セットの処理に使用するのと同じ関数を使って検索することができます。

構文

```
SQLRETURN   SQLProcedures    (SQLHSTMT         StatementHandle,
                              SQLCHAR          *CatalogName,
                              SQLSMALLINT      NameLength1,
                              SQLCHAR          *SchemaName,
                              SQLSMALLINT      NameLength2,
                              SQLCHAR          *ProcName,
                              SQLSMALLINT      NameLength3);
```

関数引き数

表 137. SQLTables の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLCHAR *	CatalogName	入力	3 分割のプロシージャー名のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	NameLength1	入力	*CatalogName* の長さ。 0 に設定してください。
SQLCHAR *	SchemaName	入力	スキーマ名で結果セットを修飾する *pattern-value* を保管する可能性のあるバッファー。 DB2 UDB for zOS および OS/390 V 4.1 の場合、すべてのストアード・プロシージャーは 1 つのスキーマになっています。 *SchemaName* 引き数に使用できる唯一の値は NULL ポインターです。 DB2 Universal Database の場合、 *SchemaName* には有効なパターン値を入れることができます。
SQLSMALLINT	NameLength2	入力	*SchemaName* の長さ。
SQLCHAR *	ProcName	入力	プロシージャー名で結果セットを修飾する *pattern-value* を保管できるバッファー。
SQLSMALLINT	NameLength3	入力	*ProcName* の長さ。

使用法

SQLProcedures() によって戻される結果セットには、 223 ページの表 138 に示された列が指定の順序で入れられます。行は、 PROCEDURE_CAT、 PROCEDURE_SCHEMA、および PROCEDURE_NAME の順になります。

多くの場合、SQLProcedures() の呼び出しは、システム・カタログに対する複雑な (そのため、経費のかさむ) 照会にマップされるので、慎重に使用する必要があり、何回も呼び出さなくて済むように結果を保管しておかなければなりません。

今後のリリースでは、新しい列が追加されたり、既存の列が変更されたりする可能性はありますが、現行列の位置は変更されません。

表 138. SQLProcedures から戻される列

1	PROCEDURE_CAT	VARCHAR(128)	現行サーバー。
2	PROCEDURE_SCHEM	VARCHAR(128)	PROCEDURE_NAME が入っているスキーマの名前。
3	PROCEDURE_NAME	VARCHAR(128) NOT NULL	プロシージャーの名前。
4	NUM_INPUT_PARAMS	NULL 以外の INTEGER	入力パラメーター数。 この列を使用しないでください。これは、 ODBC での将来の使用のために予約済みです。 これは、バージョン 5 より前の DB2 UDB CLI で使われていたものです。これは、逆方向の互換性があるので、旧 DB2CLI.PROCEDURES 疑似カタログ表と一緒に使うことができます (PATCH1 CLI/ODBC 構成キーワードを設定して)。
5	NUM_OUTPUT_PARAMS	NULL 以外の INTEGER	出力パラメーター数。 この列を使用しないでください。これは、 ODBC での将来の使用のために予約済みです。 これは、バージョン 5 より前の DB2 UDB CLI で使われていたものです。これは、逆方向の互換性があるので、旧 DB2CLI.PROCEDURES 疑似カタログ表と一緒に使うことができます (PATCH1 CLI/ODBC 構成キーワードを設定して)。
6	NUM_RESULT_SETS	NULL 以外の INTEGER	プロシージャーで戻される結果セットの数。 この列を使用しないでください。これは、 ODBC での将来の使用のために予約済みです。 これは、バージョン 5 より前の DB2 UDB CLI で使われていたものです。これは、逆方向の互換性があるので、旧 DB2CLI.PROCEDURES 疑似カタログ表と一緒に使うことができます (PATCH1 CLI/ODBC 構成キーワードを設定して)。
7	REMARKS	VARCHAR(254)	プロシージャーに関する記述情報が入ります。

注: DB2 UDB CLI で使われる列名は、X/Open CLI CAE 仕様スタイルに準拠します。列のタイプ、内容、および順序は、 ODBC において SQLProcedures() の結果セット用に定義されているものと同じです。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING

SQLProcedures

- SQL_ERROR
- SQL_INVALID_HANDLE

エラー状況

表 139. SQLProcedures SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルは、ステートメント・ハンドル上ですでにオープンしています。
40003 **08**S01	通信リンク障害	関数の完了前に、アプリケーション・プログラムとデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	DB2 UDB CLI は、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY008	操作取り消し	
HY010	関数シーケンス・エラー	
HY014	ハンドルが不足	内部リソースに起因して DB2 UDB CLI はハンドルを割り振れませんでした。
HY090	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、 SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	DB2 UDB CLI では、プロシージャー名の修飾子として *catalog* をサポートしていません。 接続先のサーバーは、プロシージャー名の修飾子としてスキーマをサポートしていません。
HYT00	タイムアウト満了	

制約事項

アプリケーション・プログラムが、ストアード・プロシージャーのカタログをサポートしない DB2 に接続されているか、またはストアード・プロシージャーをサポートしない場合は、 SQLProcedureColumns() は空の結果セットを戻します。

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 */
```

参照

- 216 ページの『SQLProcedureColumns - プロシージャーの入出力パラメーター情報の入手』

SQLPutData - パラメーターのデータ値に引き渡し

目的

SQLPutData() は、 SQLParamData() 呼び出しが SQL_NEED_DATA を戻した後にパラメーターのデータ値を提供するのに呼び出します。この関数は、大きなパラメーター値を断片的に送信する場合に使用できます。

構文

```
SQLRETURN SQLPutData  (SQLHSTMT     hstmt,
                       SQLPOINTER   rgbValue,
                       SQLINTEGER   cbValue);
```

関数引き数

表 140. SQLPutData の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLPOINTER	*rgbValue*	入力	パラメーターの実際のデータ、またはデータ部分へのポインター。データは、パラメーターの指定時にアプリケーション・プログラムが使用した SQLBindParam() 呼び出しで指定した形式になっている必要があります。
SQLINTEGER	*cbValue*	入力	*rgbValue* の長さ。 SQLPutData() への呼び出しで送信されるデータの量を指定します。 このデータ量は、特定のパラメーターの呼び出しごとに異なる可能性があります。また、アプリケーション・プログラムでは *cbValue* に SQL_NTS または SQL_NULL_DATA を指定することもできます。 日付、時刻、タイム・スタンプ・データ・タイプ、および数値データ・タイプの場合、*cbValue* は無視されます。 C バッファー・タイプが SQL_CHAR か SQL_BINARY の場合、または C バッファー・タイプとして SQL_DEFAULT が指定されており、 C バッファー・タイプのデフォルト値が SQL_CHAR か SQL_BINARY になっている場合、この値が *rgbValue* バッファーのデータのバイト数になります。

使用法

アプリケーション・プログラムは、 SQL_NEED_DATA 状態のステートメントで SQLParamData() を呼び出した後に SQLPutData() を呼び出し、 SQL_DATA_AT_EXEC パラメーターにデータ値を提供します。長いデータは、 SQLPutData() を何回か呼び出して断片的に送信できます。このパラメーターのすべてのデータ断片の送信が完了すると、アプリケーション・プログラムは、もう一度 SQLParamData() を呼び出しま

す。 SQLParamData() は、次の SQL_DATA_AT_EXEC パラメーターに進むか、または、すべてのパラメーターにデータ値が割り当てられている場合は、ステートメントを実行します。

固定長パラメーターの場合、 SQLPutData() を 2 回以上呼び出すことはできません。

入力データが文字または 2 進データである場合、 SQLPutData() 呼び出しの後に呼び出せる有効な関数呼び出しは、 SQLParamData()、SQLCancel()、または再度の SQLPutData() だけです。 SQLParamData() の場合同様、このステートメント・ハンドルを使用して呼び出す他の関数は、すべて失敗します。さらに、 *hstmt* の親 *hdbc* を参照する関数呼び出しも、その接続の属性または状態の変更に関係している場合は、すべて失敗します。 これらの関数のリストについては、 205 ページの『SQLParamData - データ値が必要な次のパラメーターの取得』 の使用法セクションを参照してください。

1 つのパラメーターで SQLPutData() を 1 回または何回か呼び出して SQL_SUCCESS が戻される場合に、 *cbValue* を SQL_NULL_DATA に設定して SQLPutData() 呼び出しを試行すると、 SQLSTATE が **HY**011 のエラーになります。このエラーが発生しても状態は変化しません。ステートメント・ハンドルは *Need Data* 状態のままなので、アプリケーション・プログラムはパラメーター・データの送信を続行できます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

以下の診断状況のうちのいくつかは、 SQLPutData() 呼び出し時ではなく、 SQLParamData() の最終呼び出し時に報告される場合があります。

表 141. SQLPutData SQLSTATE

SQLSTATE	説明	解説
22001	データが過多	SQLPutData() によって現在のパラメーターに提供されたデータのサイズは、パラメーター・サイズを超えています。 SQLPutData() の最後の呼び出しで提供したデータは無視されます。
01004	データは切り捨てられる	数値パラメーターに送信されたデータは切り捨てられましたが、有効な数字は失われませんでした。 日付または時刻の列に送信されたタイム・スタンプ・データが切り捨てられました。 この関数は SQL_SUCCESS_WITH_INFO を戻します。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	引き数 *rgbValue* は NULL ポインターです。 引き数 *rgbValue* は NULL ポインターではなく、引き数 *cbValue* は 0 未満ですが、 SQL_NTS または SQL_NULL_DATA に等しくありません。

SQLPutData

表 141. SQLPutData SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数シーケンス・エラー	ステートメント・ハンドル *hstmt* は、 **need data** 状態で、かつ直前の SQLParamData() 呼び出しで SQL_DATA_AT_EXEC パラメーターに設定されている必要があります。

SQLReleaseEnv - すべての環境リソースの解放

目的

SQLReleaseEnv() は環境ハンドルを無効にし、解放します。環境ハンドルに関連したすべての DB2 UDB CLI リソースが解放されます。

この関数より先に SQLFreeConnect() を呼び出す必要があります。

この関数が、終了処理に入る前にアプリケーション・プログラムで実行する必要のある最後の DB2 UDB CLI ステップになります。

構文

```
SQLRETURN SQLReleaseEnv (SQLHENV    henv);
```

関数引き数

表 142. SQLReleaseEnv の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル

使用法

有効な接続ハンドルがまだ存在しているのにこの関数を呼び出すと、 SQL_ERROR が戻され、環境ハンドルは有効のままになります。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 143. SQLReleaseEnv SQLSTATE

SQLSTATE	説明	解説
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY010	関数シーケンス・エラー	割り振りまたは接続状態になっている *hdbc* があります。SQLReleaseEnv を呼び出す前に、 *hdbc* に対して SQLDisconnect と SQLFreeConnect を呼び出してください。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

SQLReleaseEnv

例

31 ページの『例』SQLAllocEnv() を参照してください。

参照

- 121 ページの『SQLFreeConnect - 接続ハンドルの解放』

SQLRowCount - 行数の取得

目的

SQLRowCount() は、実行される UPDATE、INSERT、または DELETE の影響を受ける表、またはこの表に基づくビューに行数を戻します。

この関数の前に、 SQLExecute() または SQLExecDirect() を呼び出す必要があります。

構文

```
SQLRETURN SQLRowCount (SQLHSTMT      hstmt,
                       SQLINTEGER    *pcrow);
```

関数引き数

表 144. SQLRowCount の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLINTEGER *	*pcrow*	出力	影響を受ける行数が保管される場所へのポインター。

使用法

入力ステートメント・ハンドルが参照するステートメントのうち、最後に実行されるステートメントが UPDATE、INSERT、または DELETE ステートメントでない場合、または正常実行されなかった場合、*pcrow* の値はこの関数により 0 に設定されます。

このステートメントの影響を受けた他の表の任意の行 (カスケード削除など) は、この数には含まれていません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 145. SQLRowCount SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	*pcrow* が NULL ポインターです。
HY010	関数シーケンス・エラー	*hstmt* に対し、 SQLExecute または SQLExecDirect より先にこの関数が呼び出されました。

表 145. SQLRowCount SQLSTATE (続き)

SQLSTATE	説明	解説
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

参照

- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 203 ページの『SQLNumResultCols - 結果列の数の取得』

SQLSetConnectAttr - 接続属性の設定

目的

SQLSetConnectAttr() は、特定の接続の接続属性を設定します。

構文

```
SQLRETURN SQLSetConnectAttr  (SQLHDBC     hdbc,
                              SQLINTEGER  fAttr,
                              SQLPOINTER  vParam,
                              SQLINTEGER  sLen);
```

関数引き数

表 146. SQLSetConnectAttr の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル
SQLINTEGER	*fAttr*	入力	設定する接続属性。詳細については、表 147 を参照してください。
SQLPOINTER	*vParam*	入力	*fAttr* に関連する値。このオプションに応じ、32 ビット整数値、または文字ストリングになります。
SQLINTEGER	*sLen*	入力	文字ストリングの場合は入力値の長さ。その他の場合は使用されません。

使用法

SQLSetConnectAttr() で設定したすべての接続オプションおよびステートメント・オプションは、SQLFreeConnect() を呼び出すか、次に SQLSetConnectAttr() を呼び出すまで保たれます。

vParam で設定した情報形式は、指定される *fAttr* によって異なります。このオプション情報は、32 ビット整数値、または NULL 終了文字ストリングへのポインターのどちらかになります。

表 147. 接続オプション

fAttr	内容
SQL_ATTR_AUTOCOMMIT	接続のコミット動作を設定する 32 ビット値。次のような値を指定できます。 • SQL_TRUE - 各 SQL ステートメントは、実行時に自動的にコミットされます。 • SQL_FALSE - SQL ステートメントは、自動的にコミットされません。コミットメント制御を使用して実行している場合は、 SQLEndTran() または SQLTransact() を使用して、変更を明示的にコミットするかロールバックしなければなりません。

表 147. 接続オプション (続き)

fAttr	内容
SQL_ATTR_COMMIT または SQL_TXN_ISOLATION	*hdbc* が参照する現行接続のトランザクション分離レベルを設定する 32 ビット値。 DB2 UDB CLI では以下の値が受け入れられますが、個々のサーバーでサポートしている分離レベルは、このうちのいくつかに限られる場合があります。 • SQL_TXN_NO_COMMIT - コミットメント制御は使用されません。 • SQL_TXN_READ_UNCOMMITTED - ダーティー読み取り、反復不能読み取り、およびファントムは可能です。 • SQL_TXN_READ_COMMITTED - ダーティー読み取りは不可です。反復不能読み取り、およびファントムは可能です。 • SQL_TXN_REPEATABLE_READ - ダーティー読み取りおよび反復不能読み取りは不可です。ファントムは可能です。 • SQL_TXN_SERIALIZABLE - トランザクションはシリアル化可能です。ダーティー読み取り、反復不能読み取り、およびファントムは不可です。 IBM 用語に言い換えると、以下のようになります。 • SQL_TXN_READ_UNCOMMITTED は非コミット読み取り (UR)。 • SQL_TXN_READ_COMMITTED はカーソル固定 (CS)。 • SQL_TXN_REPEATABLE_READ は読み取り固定 (RS)。 • SQL_TXN_SERIALIZABLE は反復可能読み取り (RR)。 分離レベルの詳細な説明については、*「IBM SQL 解説書」*を参照してください。 SQL_ATTR_COMMIT 属性は、`SQLConnect()` の前に設定しなければなりません。接続の確立後にこの値を変更した場合に、その接続先がリモート・データ・ソースであると、接続ハンドル用の次の `SQLConnect()` が正常に完了しない限り、その変更内容は有効化されません。

表 147. 接続オプション (続き)

fAttr	内容
SQL_ATTR_DATE_FMT	32 ビット整数値。以下のいずれかになります。 • SQL_FMT_ISO - 国際標準化機構 (ISO) の日付形式 yyyy-mm-dd を使います。これがデフォルトです。 • SQL_FMT_USA - 米国日付形式 mm/dd/yyyy を使います。 • SQL_FMT_EUR - ヨーロッパ日付形式 dd.mm.yyyy を使います。 • SQL_FMT_JIS - 日本工業規格の日付形式 yyyy-mm-dd を使います。 • SQL_FMT_MDY - 日付形式 mm/dd/yyyy を使います。 • SQL_FMT_DMY - 日付形式 dd/mm/yyyy を使います。 • SQL_FMT_YMD - 日付形式 yy/mm/dd を使います。 • SQL_FMT_JUL - 年間通算日の形式 yy/ddd を使います。 • SQL_FMT_JOB - ジョブのデフォルトを使います。
SQL_ATTR_DATE_SEP	32 ビット整数値。以下のいずれかになります。 • SQL_SEP_SLASH - 斜線 (/) を日付区切り記号に使います。これがデフォルトです。 • SQL_SEP_DASH - ダッシュ (-) を日付区切り記号に使います。 • SQL_SEP_PERIOD - ピリオド (.) を日付区切り記号に使います。 • SQL_SEP_COMMA - コンマ (,) を日付区切り記号に使います。 • SQL_SEP_BLANK - ブランクを日付区切り記号に使います。 • SQL_SEP_JOB - ジョブのデフォルトを使います。
SQL_ATTR_DBC_DEFAULT_LIB	未修飾のファイル参照を解決するのに使われるデフォルト・ライブラリーを指示する文字値。システムの命名モードを使用する接続の場合、これは無効です。
SQL_ATTR_DBC_SYS_NAMING	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - DB2 UDB CLI は、iSeries システムの命名モードを使います。ファイルは、斜線区切り文字 (/) を使って修飾されます。 修飾されていないファイルは、ジョブ用のライブラリー・リストを使って解決されます。 • SQL_FALSE - DB2 UDB CLI は、デフォルトの命名モード (SQL 命名) を使います。ファイルは、ピリオド (.) 区切り文字を使って修飾されます。修飾されていないファイルは、デフォルト・ライブラリーまたは現在のユーザー ID を使って解決されます。

表 147. 接続オプション (続き)

fAttr	内容
SQL_ATTR_DECIMAL_SEP	32 ビット整数値。以下のいずれかになります。 • SQL_SEP_PERIOD - ピリオド (.) を小数点区切り記号に使います。これがデフォルトです。 • SQL_SEP_COMMA - コンマ (,) を日付区切り記号に使います。 • SQL_SEP_JOB - ジョブのデフォルトを使います。
SQL_ATTR_EXTENDED_COL_INFO	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - この接続ハンドルに対して割り振られるステートメント・ハンドルを SQLColAttributes() で使用して、基本表、基本スキーマ、基本列、およびラベルなどの拡張された列情報を検索できます。 • SQL_FALSE - この接続ハンドルに対して割り振られるステートメント・ハンドルを SQLColAttributes() 関数で使用して、拡張された列情報を検索することはできません。これがデフォルトです。
SQL_ATTR_TIME_FMT	32 ビット整数値。以下のいずれかになります。 • SQL_FMT_ISO - 国際標準化機構 (ISO) の時刻形式 hh.mm.ss を使います。これがデフォルトです。 • SQL_FMT_USA - 米国時刻形式 hh:mmxx を使います。xx は AM または PM です。 • SQL_FMT_EUR - ヨーロッパの時刻形式 hh.mm.ss を使います。 • SQL_FMT_JIS - 日本工業規格の時刻形式 hh:mm:ss を使います。 • SQL_FMT_HMS - hh:mm:ss 形式を使います。
SQL_ATTR_TIME_SEP	32 ビット整数値。以下のいずれかになります。 • SQL_SEP_COLON - コロン (:) を時刻区切り記号に使います。これがデフォルトです。 • SQL_SEP_PERIOD - ピリオド (.) を時刻区切り記号に使います。 • SQL_SEP_COMMA - コンマ (,) を時刻区切り記号に使います。 • SQL_SEP_BLANK - ブランクを時刻区切り記号に使います。 • SQL_SEP_JOB - ジョブのデフォルトを使います。
SQL_SAVEPOINT_NAME	関数 SQL_SAVEPOINT_NAME_ROLLBACK または SQL_SAVEPOINT_NAME_RELEASE 上で SQLEndTran() によって使用される、保管点の名前を示す文字値。

表 147. 接続オプション (続き)

fAttr	内容
SQL_2ND_LEVEL_TEXT	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - SQLError() を呼び出して入手するエラー・テキストに、エラーの完全なテキスト記述が含まれることになります。 • SQL_FALSE - SQLError() を呼び出して入手するエラー・テキストに、エラーの第 1 レベルの説明だけが含まれることになります。これがデフォルトです。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 148. SQLSetConnectAttr SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	指定された *fAttr* 値が与えられましたが、 *vParam* の引き数に指定された値が無効です。 指定された *fAttr* 値は無効です。

SQLSetConnectOption - 接続オプションの設定

目的

SQLSetConnectOption() は、特定の接続の接続属性を設定します。

構文

```
SQLRETURN SQLSetConnectOption  (SQLHDBC hdbc,
                                SQLSMALLINT fOption,
                                SQLPOINTER  vParam);
```

関数引き数

表 149. SQLSetConnectOption の引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*hdbc*	入力	接続ハンドル
SQLSMALLINT	*fOption*	入力	設定する接続オプション。詳細については、233 ページの表 147 を参照してください。
SQLPOINTER	*vParam*	入力	*fOption* に関連する値。このオプションに応じ、32 ビット整数値、または文字ストリングになります。

使用法

SQLSetConnectOption() は、 SQLSetConnectAttr() と同じ関数を提供していますが、どちらの関数も互換性の理由でサポートされています。

SQLSetConnectOption() で設定したすべての接続オプションおよびステートメント・オプションは、SQLFreeConnect() を呼び出すか、次に SQLSetConnectOption() を呼び出すまで保たれます。

vParam で設定した情報形式は、指定される *fOption* によって異なります。このオプション情報は、32 ビット整数値、または NULL 終了文字ストリングへのポインターのどちらかになります。

適切な接続オプションについては、233 ページの表 147 を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 150. SQLSetConnectOption SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 150. SQLSetConnectOption SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数値が無効	指定された *fOption* 値が与えられましたが、 *vParam* の引き数に指定された値が無効です。 指定された *fOption* 値は無効です。
HYC00	ドライバーでサポートされていない	指定された *fOption* は、DB2 UDB CLI でもサーバーでもサポートされていません。 指定された *fOption* 値が与えられましたが、引き数 *vParam* に指定されている値はサポートされていません。

SQLSetCursorName - カーソル名の設定

目的

SQLSetCursorName() は、カーソル名をステートメント・ハンドルに関連付けます。 DB2 UDB CLI では必要に応じて暗黙的にカーソル名を生成するので、この関数は任意指定です。

構文

```
SQLRETURN SQLSetCursorName (SQLHSTMT       hstmt,
                            SQLCHAR        *szCursor,
                            SQLSMALLINT    cbCursor);
```

関数引き数

表 151. SQLSetCursorName の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLCHAR *	*szCursor*	入力	カーソル名
SQLSMALLINT	*cbCursor*	入力	*szCursor* 引き数の内容の長さ

使用法

DB2 UDB CLI では、SELECT ステートメントの準備作成または直接実行時に、常に内部生成カーソル名を使用します。 SQLSetCursorName() を使用すると、 SQL ステートメント (位置の決まった UPDATE または DELETE) でアプリケーション・プログラム定義のカーソル名を使用できるようになります。 DB2 UDB CLI は、この名前を内部名にマップします。SQLSetCursorName() は、内部名が生成される前に呼び出す必要があります。この名前は、ハンドルがドロップされるまでステートメント・ハンドルと関連付けられたままになります。また、この名前はトランザクション終了後も残りますが、この時点で SQLSetCursorName() を呼び出して、このステートメント・ハンドルに異なる名前を設定することもできます。

カーソル名に関する規則は、以下のとおりです。

- 接続内のすべてのカーソル名は、固有でなければならない。
- それぞれのカーソル名の長さは、18 バイト以下でなければならない。 18 バイトを超える長さのカーソル名を設定しようとすると、このカーソル名は 18 バイトで切り捨てられます。 (警告は生成されません。)
- SQL ではカーソル名を ID と見なすので、先頭は英字 (a〜z、A〜Z)、その後は数字 (0〜9)、英字、または下線文字 (_) の任意の組み合わせになっていなければならない。
- 入力カーソル名を二重引用符で囲まないと、入力カーソル名ストリングのすべての先行ブランクおよび後書きブランクは、削除されます。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 152. SQLSetCursorName SQLSTATE

SQLSTATE	説明	解説
34000	カーソル名が無効	引き数 *szCursor* に指定されたカーソル名が無効です。カーソル名が ″SQLCUR″ または ″SQL_CUR″ で始まっているか、ドライバーまたはデータ・ソース・カーソルの命名規則 (先頭は英字 (a〜z、A〜Z)、その後は数字 (0〜9)、英字、または下線文字 (_) の任意の組み合わせ) に違反しています。 引き数 *szCursor* に指定されたカーソル名は存在しています。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	*szCursor* が NULL ポインターになっていました。 引き数 *cbCursor* は 1 未満でしたが、 SQL_NTS と同じになっていませんでした。
HY010	関数シーケンス・エラー	ステートメント・ハンドルが割り振り状態になっていませんでした。 `SQLPrepare()` または `SQLExecDirect()` が、`SQLSetCursorName()` より先に呼び出されました。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

参照

- 139 ページの『SQLGetCursorName - カーソル名の取得』

SQLSetDescField - 記述子フィールドの設定

目的

SQLSetDescField() は、記述子のフィールドを設定します。 SQLSetDescField() は、 SQLSetDescRec() 関数を拡張した代替関数です。

構文

```
SQLRETURN SQLSetDescField  (SQLHDESC       hdesc,
                            SQLSMALLINT    irec,
                            SQLSMALLINT    fDescType,
                            SQLPOINTER     rgbDesc,
                            SQLINTEGER     bLen);
```

関数引き数

表 153. SQLSetDescField の引き数

データ・タイプ	引き数	使用法	説明
SQLHDESC	*hdesc*	入力	記述子ハンドル
SQLSMALLINT	*irec*	入力	指定されたフィールドを検索するレコード番号。
SQLSMALLINT	*fDescType*	入力	表 154 を参照してください。
SQLPOINTER	*rgbDesc*	入力	バッファーへのポインター。
SQLINTEGER	*bLen*	入力	記述子バッファーの長さ (*rgbDesc*)

表 154. fDescType 記述子タイプ

記述子	タイプ	説明
SQL_DESC_COUNT	SMALLINT	記述子のレコード数を設定。 *irec* は無視されます。
SQL_DESC_TYPE	SMALLINT	*irec* のタイプ・フィールドを設定。
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	SQL_DATETIME タイプのレコードに時間間隔コードを設定。
SQL_DESC_LENGTH	INTEGER	*irec* の長さフィールドを設定。
SQL_DESC_PRECISION	SMALLINT	*irec* の精度フィールドを設定。
SQL_DESC_SCALE	SMALLINT	*irec* の位取りフィールドを設定。
SQL_DESC_DATA_PTR	SQLPOINTER	*irec* のポインター・フィールドを設定。
SQL_DESC_LENGTH_PTR	SQLPOINTER	*irec* の長さポインター・フィールドを設定。
SQL_DESC_INDICATOR_PTR	SQLPOINTER	*irec* のインディケーター・ポインター・フィールドを設定。

使用法

SQLSetDescField() は、引き数セット全体が必要になる SQLSetDescRec() とは異なり、特定の記述子レコードに設定したい属性を指定します。

SQLSetDescField() の将来的な拡張は可能ですが、 SQLSetDescRec() に比べ、それぞれの記述子レコードに同じ情報を設定するのにより多くの呼び出しが必要になります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 155. SQLGetDescField SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *fDescType* または *irec* に指定された値が有効ではありませんでした。 引き数 *rgbValue* は NULL ポインターです。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLSetDescRec - 記述子レコードの設定

目的

SQLSetDescRec() は、記述子レコードのすべての属性を設定します。SQLSetDescRec() は、SQLDescField() 関数のより簡素な代替関数として使用できます。

構文

```
SQLRETURN SQLSetDescRec (SQLHDESC       hdesc,
                         SQLSMALLINT    irec,
                         SQLSMALLINT    type,
                         SQLSMALLINT    subtype,
                         SQLINTEGER     length,
                         SQLSMALLINT    prec,
                         SQLSMALLINT    scale,
                         SQLPOINTER     data,
                         SQLINTEGER     *sLen,
                         SQLINTEGER    *indic);
```

関数引き数

表 156. SQLSetDescRec の引き数

データ・タイプ	引き数	使用法	説明
SQLDESC	*hdesc*	入力	記述子ハンドル
SQLSMALLINT	*irec*	入力	記述子内のレコード番号。
SQLSMALLINT	*type*	入力	レコードの TYPE フィールド。
SQLSMALLINT	*subtype*	入力	TYPE が SQL_DATETIME になっているレコードの場合は DATETIME_INTERVAL_CODE フィールド。
SQLINTEGER	*length*	入力	レコードの LENGTH フィールド。
SQLSMALLINT	*prec*	入力	レコードの PRECISION フィールド。
SQLSMALLINT	*scale*	入力	レコードの SCALE フィールド。
SQLPOINTER	*data*	入力 (据え置き)	レコードの DATA_PTR フィールド。
SQLINTEGER *	*sLen*	入力 (据え置き)	レコードの LENGTH_PTR フィールド。
SQLINTEGER *	*indic*	入力 (据え置き)	レコードの INDICATOR_PTR フィールド。

使用法

SQLSetDescRec() を呼び出すと、記述子レコードのすべてのフィールドを 1 回の呼び出しで設定できます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 157. SQLSetDescRec SQLSTATE

SQLSTATE	説明	解説
HY009	引き数値が無効	引き数 *irec* に指定された値が、1 未満になっていました。 別の引き数に無効な値が指定されました。
HY016	記述子が無効	記述子ハンドルが実装の行記述子を参照しました。

参照

- 37 ページの『SQLBindCol - アプリケーション・プログラム変数に対する列のバインド』
- 83 ページの『SQLDescribeCol - 列属性の記述』
- 101 ページの『SQLExecDirect - ステートメントの直接実行』
- 103 ページの『SQLExecute - ステートメントの実行』
- 209 ページの『SQLPrepare - ステートメントの準備作成』

SQLSetEnvAttr - 環境属性の設定

目的

SQLSetEnvAttr() は、現在の環境の環境属性を設定します。

構文

```
SQLRETURN SQLSetEnvAttr  (SQLHENV      henv,
                          SQLINTEGER   Attribute,
                          SQLPOINTER   Value,
                          SQLINTEGER   StringLength);
```

関数引き数

表 158. SQLSetEnvAttr の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル
SQLINTEGER	*Attribute*	入力	設定する環境属性。詳細については、 表 159 を参照してください。
SQLPOINTER	*pValue*	入力	*Attribute* に使用したい値。
SQLINTEGER	*StringLength*	入力	属性値が文字ストリングの場合、バイト単位の *Value* の長さ。 *Attribute* がストリングでない場合、 DB2 UDB CLI は *StringLength* を無視します。接続ハンドルであってはなりません。そうでないと、エラー HY010 になります。

使用法

表 159. 環境属性

Attribute	内容
SQL_ATTR_DATE_FMT	32 ビット整数値。以下のいずれかになります。 • SQL_FMT_ISO - 国際標準化機構 (ISO) の日付形式 yyyy-mm-dd を使います。これがデフォルトです。 • SQL_FMT_USA - 米国日付形式 mm/dd/yyyy を使います。 • SQL_FMT_EUR - ヨーロッパ日付形式 dd.mm.yyyy を使います。 • SQL_FMT_JIS - 日本工業規格の日付形式 yyyy-mm-dd を使います。 • SQL_FMT_MDY - 日付形式 mm/dd/yyyy を使います。 • SQL_FMT_DMY - 日付形式 dd/mm/yyyy を使います。 • SQL_FMT_YMD - 日付形式 yy/mm/dd を使います。 • SQL_FMT_JUL - 年間通算日の形式 yy/ddd を使います。 • SQL_FMT_JOB - ジョブのデフォルトを使います。

表 159. 環境属性 (続き)

Attribute	内容
SQL_ATTR_DATE_SEP	32 ビット整数値。以下のいずれかになります。 • SQL_SEP_SLASH - 斜線 (/) を日付区切り記号に使います。これがデフォルトです。 • SQL_SEP_DASH - ダッシュ (-) を日付区切り記号に使います。 • SQL_SEP_PERIOD - ピリオド (.) を日付区切り記号に使います。 • SQL_SEP_COMMA - コンマ (,) を日付区切り記号に使います。 • SQL_SEP_BLANK - ブランクを日付区切り記号に使います。 • SQL_SEP_JOB - ジョブのデフォルトを使います。
SQL_ATTR_DECIMAL_SEP	32 ビット整数値。以下のいずれかになります。 • SQL_SEP_PERIOD - ピリオド (.) を小数点区切り記号に使います。これがデフォルトです。 • SQL_SEP_COMMA - コンマ (,) を日付区切り記号に使います。 • SQL_SEP_JOB - ジョブのデフォルトを使います。
SQL_ATTR_DEFAULT_LIB	未修飾のファイル参照を解決するのに使われるデフォルト・ライブラリーを指示する文字値。システムの命名モードを使用する環境の場合、これは無効です。
SQL_ATTR_ENVHNDL_COUNTER	32 ビット整数値。以下のどちらかになります。 • SQL_FALSE - DB2 CLI は、環境ハンドルが割り振られた回数をカウントしません。したがって、環境ハンドルを解放するための最初の呼び出しによって、ハンドルとそれに関連したすべてのリソースが解放されます。 • SQL_TRUE - DB2 CLI は、環境ハンドルが割り振られた回数のカウンターを保存します。環境ハンドルの解放ごとに、カウンターは減ります。カウンターがゼロに達して初めて DB2 CLI は、ハンドルとそれに関連したリソースを実際に解放します。そのため、CLI を使ってプログラムに対して、 CLI 環境ハンドルの割り振りと解放のためのネストされた呼び出しを行うことができます。
SQL_ATTR_ESCAPE_CHAR	SQLColumns( ) または SQLTables( ) に検索パターンを指定するのに使用するエスケープ文字を指示する文字値。

表 159. 環境属性 (続き)

Attribute	内容
SQL_ATTR_FOR_FETCH_ONLY	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - カーソルは読み取り専用で、位置の決まった更新または削除には使用できない。 これがデフォルトです。 • SQL_FALSE - カーソルを、位置の決まった更新および削除に使用できる。 また、SQLSetStmtAttr() を使って、個々のステートメントごとに属性 SQL_ATTR_FOR_FETCH_ONLY を設定することもできます。
SQL_ATTR_JOB_SORT_SEQUENCE	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - DB2 UDB CLI は、ジョブ用に設定されているソート・シーケンスを使います。 • SQL_FALSE - DB2 UDB CLI は、デフォルトのソート・シーケンス (*HEX) を使います。
SQL_ATTR_OUTPUT_NTS	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - DB2 UDB CLI は、NULL 終了文字を使用して、出力文字ストリングの長さを指示します。 • SQL_FALSE - DB2 UDB CLI は NULL 終了文字を使用しません。 この環境 (およびこの環境で割り振られたすべての接続) に呼び出される CLI 関数のうち、文字ストリング・パラメーターを持つすべての CLI 関数は、この属性の影響を受けます。
SQL_ATTR_SERVER_MODE	32 ビット整数値。以下のどちらかになります。 • SQL_FALSE - DB2 CLI は、同じジョブ内のすべての接続の SQL ステートメントを処理します。 すべての変更が、1 つのトランザクションを構成します。これが、デフォルトの処理モードです。 • SQL_TRUE - DB2 CLI は、別のジョブ内の各接続の SQL ステートメントを処理します。すると、同じデータ・ソースに対して、それぞれの接続ごとに別々のユーザー ID で、複数の接続を確立することができます。また、それぞれの接続ハンドルで行われた変更を分離して、自身のトランザクションに入れます。すると、他の接続ハンドルのもとで行われた保留中の変更内容に影響を与えずに、各接続ハンドルをコミットまたはロールバックさせることができます。詳細については、 303 ページの『付録 D. サーバー・モードでの DB2 UDB CLI の実行』を参照してください。

表 159. 環境属性 (続き)

Attribute	内容
SQL_ATTR_SYS_NAMING	32 ビット整数値。以下のどちらかになります。 • SQL_TRUE - DB2 UDB CLI は、iSeries システムの命名モードを使います。ファイルは、斜線区切り文字 (/) を使って修飾されます。 修飾されていないファイルは、ジョブ用のライブラリー・リストを使って解決されます。 • SQL_FALSE - DB2 UDB CLI は、デフォルトの命名モード (SQL 命名) を使います。ファイルは、ピリオド (.) 区切り文字を使って修飾されます。修飾されていないファイルは、デフォルト・ライブラリーまたは現在のユーザー ID を使用して解決されます。
SQL_ATTR_TIME_FMT	32 ビット整数値。以下のいずれかになります。 • SQL_FMT_ISO - 国際標準化機構 (ISO) の時刻形式 hh.mm.ss を使います。これがデフォルトです。 • SQL_FMT_USA - 米国時刻形式 hh:mmxx を使います。xx は AM または PM です。 • SQL_FMT_EUR - ヨーロッパの時刻形式 hh.mm.ss を使います。 • SQL_FMT_JIS - 日本工業規格の時刻形式 hh:mm:ss を使います。 • SQL_FMT_HMS - hh:mm:ss 形式を使います。
SQL_ATTR_TIME_SEP	32 ビット整数値。以下のいずれかになります。 • SQL_SEP_COLON - コロン (:) を時刻区切り記号に使います。これがデフォルトです。 • SQL_SEP_PERIOD - ピリオド (.) を時刻区切り記号に使います。 • SQL_SEP_COMMA - コンマ (,) を時刻区切り記号に使います。 • SQL_SEP_BLANK - ブランクを時刻区切り記号に使います。 • SQL_SEP_JOB - ジョブのデフォルトを使います。
SQL_ATTR_UTF8	32 ビット整数値。以下のどちらかになります。 • SQL_FALSE - 文字データは、デフォルト・ジョブの CCSID にあるものとして処理されます。これがデフォルトです。 • SQL_TRUE - 文字データは、UTF-8 CCSID (1208) にあるものとして処理されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 160. SQLSetEnvAttr SQLSTATE

SQLSTATE	説明	解説
HY009	パラメーター値が無効	指定された *Attribute* は DB2 UDB CLI でサポートされていません。 指定された *Attribute* 値が与えられましたが、引き数 *Value* に指定されている値はサポートされていません。 引き数 *pValue* が NULL ポインターになっていました。
HY010	関数シーケンス・エラー	接続ハンドルがすでに割り振られています。

SQLSetParam - パラメーターの設定

目的

SQLSetParam() は、アプリケーション・プログラム変数を SQL ステートメントのパラメーター・マーカーに関連付け (バインド) ます。バインド変数の内容は、ステートメントの実行時にデータベース・サーバーに送信されます。また、この関数は、必要な任意のデータ変換を指定する場合にも使用されます。

構文

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

注: この関数の説明については、48 ページの『SQLBindParam - パラメーター・マーカーに対するバッファーのバインド』を参照してください。これらの関数は同じものですが、互換性の理由でサポートされています。

SQLSetStmtAttr - ステートメント属性の設定

目的

SQLSetStmtAttr() は、特定のステートメント・ハンドルの属性を設定します。接続ハンドルと関連するすべてのステートメント・ハンドルのオプションを設定する場合は、アプリケーション・プログラムから SQLSetConnectOption() を呼び出すことができます。 (詳細については、 238 ページの『SQLSetConnectOption - 接続オプションの設定』も参照してください。)

構文

```
SQLRETURN SQLSetStmtAttr  (SQLHSTMT       hstmt,
                           SQLINTEGER     fAttr,
                           SQLPOINTER     vParam,
                           SQLINTEGER     sLen);
```

関数引き数

表 161. SQLSetStmtAttr の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLINTEGER	*fAttr*	入力	設定する属性。設定可能なステートメント属性のリストについては、 表 162 を参照してください。
SQLPOINTER	*vParam*	入力	*fAttr* に関連する値。*vParam* は、 2 ビット整数値、または文字ストリングになります。
SQLINTEGER	*sLen*	入力	データが文字ストリングの場合、データの長さ。その他の場合は使用されません。

使用法

hstmt のステートメント・オプションは、もう一度 SQLSetStmtAttr() が呼び出されて変更されるか、SQL_DROP オプションを指定した SQLFreeStmt() により *hstmt* がドロップされるまで有効です。SQL_CLOSE、SQL_UNBIND、または SQL_RESET_PARAMS オプションを指定して SQLFreeStmt() を呼び出しても、ステートメント・オプションはリセットされません。

vParam で設定した情報形式は、指定される *fOption* によって異なります。それぞれの形式については、表 162 に記述されています。

表 162. ステートメント属性

fAttr	内容
SQL_ATTR_APP_PARAM_DESC	*VParam* は、記述子ハンドルでなければなりません。指定される記述子は、後でステートメント・ハンドル上で SQLExecute() および SQLExecDirect() を呼び出す際にアプリケーション・プログラム・パラメーター記述子として機能します。
SQL_ATTR_APP_ROW_DESC	*VParam* は、記述子ハンドルでなければなりません。指定される記述子ハンドルは、後でステートメント・ハンドル上で SQLFetch() を呼び出す際にアプリケーション・プログラム行記述子として機能します。

表 162. ステートメント属性 (続き)

fAttr	内容
SQL_ATTR_CURSOR_HOLD	このステートメント・ハンドルのオープンされたカーソルを保持するかどうかを指定する 32 ビット整数値。 • SQL_FALSE - このステートメント・ハンドルのオープン・カーソルは、コミットまたはロールバック操作が行われるときにクローズされます。これがデフォルトです。 • SQL_TRUE - このステートメント・ハンドルのオープン・カーソルは、コミットまたはロールバック操作が行われるときにクローズされません。
SQL_ATTR_CURSOR_SCROLLABLE	このステートメント・ハンドルのオープンされたカーソルをスクロール可能にするかどうかを指定する 32 ビット整数値。 • SQL_FALSE - カーソルをスクロール可能にしない。また、カーソルに対して SQLFetchScroll() を使用しない。これがデフォルトです。 • SQL_TRUE - カーソルをスクロール可能にする。これらのカーソルのデータ検索に、 SQLFetchScroll() を使用できます。
SQL_ATTR_CURSOR_TYPE	このステートメント・ハンドルに対してオープンされたカーソルの動作を指定する 32 ビット整数値。 • SQL_CURSOR_FORWARD_ONLY - カーソルをスクロール可能にしない。また、カーソルに対して SQLFetchScroll() を使用しない。これがデフォルトです。 • SQL_DYNAMIC - カーソルをスクロール可能にする。これらのカーソルのデータ検索に、 SQLFetchScroll() を使用できます。
SQL_ATTR_EXTENDED_COL_INFO	このステートメント・ハンドルのオープンされたカーソルが、拡張された列情報を提供するかどうかを指定する 32 ビット整数値。 • SQL_FALSE - このステートメント・ハンドルを SQLColAttributes() 関数で使用して、拡張された列情報を検索することはできません。これがデフォルトです。この属性をステートメント・レベルで設定すると、接続レベルでの属性の設定はオーバーライドされます。 • SQL_TRUE - このステートメント・ハンドルを SQLColAttributes() で使用して、基本表、基本スキーマ、基本列、およびラベルなどの拡張された列情報を検索できます。

表 162. ステートメント属性 (続き)

fAttr	内容
SQL_ATTR_FOR_FETCH_ONLY	このステートメント・ハンドルのオープンされたカーソルを読み取り専用にするかどうかを指定する 32 ビット整数値。 • SQL_TRUE - カーソルは読み取り専用で、位置の決まった更新または削除には使用できない。SQL_ATTR_FOR_FETCH_ONLY 環境を SQL_FALSE に設定していない場合、これがデフォルトになります。 • SQL_FALSE - カーソルを、位置の決まった更新および削除に使用できる。
SQL_ATTR_FULL_OPEN	このステートメント・ハンドルのオープンされたカーソルを完全にオープンするかどうかを指定する 32 ビット整数値。 • SQL_FALSE - このステートメント・ハンドルのためにカーソルをオープンすると、パフォーマンス上の理由で、キャッシュに入れられたカーソルが使用されます。これがデフォルトです。 • SQL_TRUE - このステートメント・ハンドルのためにカーソルをオープンすると、必ず新しいカーソルの完全なオープンが強制的に実行されます。
SQL_ATTR_ROWSET_SIZE	行セット内の行数を指定する 32 ビット整数値。これは、SQLExtendedFetch() の各呼び出しで戻される行数です。デフォルト値は 1 です。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 163. SQLStmtAttr SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
HY000	一般エラー	特定の SQLSTATE がなく、実装定義の SQLSTATE が定義されていないエラーが発生しました。このエラーおよび原因については、 SQLError が引き数 *szErrorMsg* に戻すエラー・メッセージに説明されています。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。

表 163. SQLStmtAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数値が無効	指定の *fAttr* 値を提供しましたが、引き数 vParam に指定した値が無効です。 指定された *fAttr* 値は無効です。 引き数 *vParam* は NULL ポインターです。
HY010	関数シーケンス・エラー	関数が、順序外で呼び出されました。
HYC00	ドライバーでサポートされていない	ドライバーまたはデータ・ソースでは、指定されたオプションがサポートされていません。

SQLSetStmtOption - ステートメント・オプションの設定

目的

SQLSetStmtOption() は、特定のステートメント・ハンドルの属性を設定します。接続ハンドルと関連するすべてのステートメント・ハンドルのオプションを設定する場合は、アプリケーション・プログラムから SQLSetConnectOption() を呼び出すことができます。 (詳細については、 238 ページの『SQLSetConnectOption - 接続オプションの設定』も参照してください。)

構文

```
SQLRETURN SQLSetStmtOption  (SQLHSTMT       hstmt,
                             SQLSMALLINT    fOption,
                             SQLPOINTER     vParam);
```

関数引き数

表 164. SQLSetStmtOption の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*fOption*	入力	設定するオプション。設定可能なステートメント・オプションのリストについては、 252 ページの表 162 を参照してください。
SQLPOINTER	*vParam*	入力	*fOption* に関連する値。 *vParam* は、 32 ビット整数値へのポインター、または文字ストリングになります。

使用法

SQLSetStmtOption() は、 SQLSetStmtAttr() と同じ関数を提供していますが、どちらの関数も互換性の理由でサポートされています。

hstmt のステートメント・オプションは、もう一度 SQLSetStmtOption() が呼び出されて変更されるか、SQL_DROP オプションを指定した SQLFreeStmt() により *hstmt* がドロップされるまで有効です。SQL_CLOSE、SQL_UNBIND、または SQL_RESET_PARAMS オプションを指定して SQLFreeStmt() を呼び出しても、ステートメント・オプションはリセットされません。

vParam で設定した情報形式は、指定される *fOption* によって異なります。それぞれの形式については、252 ページの表 162 に記述されています。

適切なステートメント・オプションについては、 252 ページの表 162 を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 165. SQLStmtOption SQLSTATE

SQLSTATE	説明	解説
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
HY000	一般エラー	特定の SQLSTATE がなく、実装定義の SQLSTATE が定義されていないエラーが発生しました。このエラーおよび原因については、 SQLError が引き数 *szErrorMsg* に戻すエラー・メッセージに説明されています。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数値が無効	指定された *fOption* 値が与えられましたが、vParam の引き数に指定された値が無効です。 指定された *fOption* 値は無効です。 引き数 *szSchemaName* または *szTableName* は NULL ポインターです。
HY010	関数シーケンス・エラー	関数が、順序外で呼び出されました。
HYC00	ドライバーでサポートされていない	ドライバーまたはデータ・ソースでは、指定されたオプションがサポートされていません。

SQLSpecialColumns - 特殊な列 (行 ID) の取得

目的

SQLSpecialColumns() は、表の固有な行 ID 情報を戻します。この情報には、固有索引または基本キー情報などが含まれます。この情報は SQL 結果セットに戻されますが、このセットは SELECT ステートメントで生成された結果セットの取り出しに使用した関数と同じ関数で検索できるようになっています。

構文

```
SQLRETURN SQLSpecialColumns  (SQLHSTMT       hstmt,
                              SQLSMALLINT    fColType,
                              SQLCHAR        *szCatalogName,
                              SQLSMALLINT    cbCatalogName,
                              SQLCHAR        *szSchemaName,
                              SQLSMALLINT    cbSchemaName,
                              SQLCHAR        *szTableName,
                              SQLSMALLINT    cbTableName,
                              SQLSMALLINT    fScope,
                              SQLSMALLINT    fNullable);
```

関数引き数

表 166. SQLSpecialColumns の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLSMALLINT	*fColType*	入力	今後、特殊な列タイプを追加してサポートするときに使用するため予約されています。 このデータ・タイプは現在は無視されています。
SQLCHAR *	*szCatalogName*	入力	3 つの部分で構成される表の名前のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	*cbCatalogName*	入力	*szCatalogName* の長さ。 0 に設定してください。
SQLCHAR *	*szSchemaName*	入力	指定された表のスキーマ修飾子。
SQLSMALLINT	*cbSchemaName*	入力	*szSchemaName* の長さ。
SQLCHAR *	*szTableName*	入力	表名。
SQLSMALLINT	*cbTableName*	入力	*cbTableName* の長さ。

表 166. SQLSpecialColumns の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*fScope*	入力	固有な行 ID の有効性が保持される必要のある期間の最小値。 *fScope* の値は、以下のうちのいずれかでなければなりません。 • SQL_SCOPE_CURROW - この行 ID の有効性が保証されるのは、その行にある間だけです。行が別のトランザクションにより更新または削除された場合は、後で同じ行 ID の値を使用して選択し直してもその行は戻されません。 • SQL_SCOPE_TRANSACTION - この行 ID の有効性は、現行トランザクションの持続期間中は保証されます。 • SQL_SCOPE_SESSION - この行 ID の有効性は、接続の持続期間中は保証されます。 行 ID の値の有効性が保証される持続期間は、現行トランザクションの分離レベルによって異なります。分離レベルに関する詳細およびシナリオについては、「*IBM DB2 SQL 解説書*」 を参照してください。
SQLSMALLINT	*fNullable*	入力	NULL 値が入っているような特殊な列を戻すかどうか判別します。 以下のいずれかでなければなりません。 • SQL_NO_NULLS 戻される行 ID 列のセットに NULL 値を含めることはできません。 • SQL_NULLABLE 戻される行 ID 列のセットに NULL 値が許可されている列を含めることはできます。

使用法

表の中の行を固有に識別する方法が何通りかある場合 (指定された表に複数の固有索引が存在しているなど)、 DB2 UDB CLI は内部規準に基づいて設定された行 ID 列のうち、 *最も適切な* セットを戻します。

表の中の行を固有に識別できるような列セットがない場合は、空の結果セットが戻されます。

固有な行 ID の情報は、行 ID の各列が結果セットの中の 1 つの行で表される結果セットの形で戻されます。 SQLSpecialColumns() が戻す結果セットには、以下の順序で以下のような列が含まれています。

SQLSpecialColumns

表 167. SQLSpecialColumns によって戻される列

列名	データ・タイプ	説明
SCOPE	NULL 以外の SMALLINT	rowid の実際の有効範囲。以下の値のうちのいずれかになります。 • SQL_SCOPE_CURROW • SQL_SCOPE_TRANSACTION • SQL_SCOPE_SESSION それぞれの値の説明については、 258 ページの表 166 の *fScope* を参照してください。
COLUMN_NAME	NULL 以外の VARCHAR(128)	行 ID の列の名前。
DATA_TYPE	NULL 以外の SMALLINT	列の SQL データ・タイプ。
TYPE_NAME	NULL 以外の VARCHAR(128)	DATA_TYPE 列値に関連する名前の DBMS 表記文字ストリング。
LENGTH_PRECISION	INTEGER	列の精度。精度が適当でないデータ・タイプの場合は、NULL が戻されます。
BUFFER_LENGTH	INTEGER	デフォルト C タイプに戻されるデータのバイト単位の長さ。 CHAR データ・タイプの場合、この値は LENGTH_PRECISION 列の値と同じになります。
SCALE	SMALLINT	列の位取り。位取りが該当しないデータ・タイプの場合は、 NULL が戻されます。
PSEUDO_COLUMN	SMALLINT	列を疑似列にするかどうか指示します。DB2 UDB CLI が戻すのは、以下の値だけです。 • SQL_PC_NOT_PSEUDO

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 168. SQLSpecialColumns SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルに関する情報を要求しましたが、オープンされたカーソルはありません。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。

表 168. SQLSpecialColumns SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数の長さが無効	長さ引き数のうち 1 つの値は 0 未満でしたが、SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	データ・ソースでは、 3 つの部分で構成される表名の *catalog* (先頭) 部分はサポートされていません。

SQLStatistics - 基本表の索引情報と統計情報の取得

目的

SQLStatistics() は、与えられた表の索引情報を検索します。また、基数、表に関連するページの数、および表の索引も戻します。この情報は結果セットに戻されますが、このセットは SELECT ステートメントで生成された結果セットの取り出しに使用した関数と同じ関数で検索できるようになっています。

構文

```
SQLRETURN SQLStatistics  (SQLHSTMT       hstmt,
                          SQLCHAR        *szCatalogName,
                          SQLSMALLINT    cbCatalogName,
                          SQLCHAR        *szSchemaName,
                          SQLSMALLINT    cbSchemaName,
                          SQLCHAR        *szTableName,
                          SQLSMALLINT    cbTableName,
                          SQLSMALLINT    fUnique,
                          SQLSMALLINT    fAccuracy);
```

関数引き数

表 169. SQLStatistics の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLCHAR *	*szCatalogName*	入力	3 つの部分で構成される表の名前のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	*cbCatalogName*	入力	*cbCatalogName* の長さ。 0 に設定してください。
SQLCHAR *	*szSchemaName*	入力	指定された表のスキーマ修飾子。
SQLSMALLINT	*cbSchemaName*	入力	*szSchemaName* の長さ。
SQLCHAR *	*szTableName*	入力	表名。
SQLSMALLINT	*cbTableName*	入力	*cbTableName* の長さ。
SQLSMALLINT	*fUnique*	入力	戻す索引情報のタイプ。 • SQL_INDEX_UNIQUE 戻されるのは、固有索引だけです。 • SQL_INDEX_ALL すべての索引が戻されます。
SQLSMALLINT	*fAccuracy*	入力	現在使用されていないので、0 に設定してください。

使用法

SQLStatistics() は、次のタイプの情報を戻します。

- 表の統計情報 (使用可能な場合)。
  - 以下の表の TYPE 列が SQL_TABLE_STAT に設定されている場合は、表の中の行数とその表の保管に使用したページ数。

– TYPE 列に索引が指示されている場合は、索引の中の固有値の数、およびその索引の保管に使用したページ数。
– それぞれの索引に関する情報。この場合、個々の索引の列は、結果セットの 1 行で表されます。結果セットの列は、以下の表に示す順序で与えられます。結果セットの行は、 NON_UNIQUE、 TYPE、 INDEX_QUALIFIER、 INDEX_QUALIFIER、 INDEX_NAME、および ORDINAL_POSITION によって順序付けされます。

表 170. SQLStatistics によって戻される列

列名	データ・タイプ	説明
TABLE_CAT	VARCHAR(128)	TABLE_SCHEM を含むカタログの名前。 NULL に設定されます。
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME が入っているスキーマの名前。
TABLE_NAME	NULL 以外の VARCHAR(128)	表の名前。
NON_UNIQUE	SMALLINT	索引で重複値を禁止するかどうかを指示します。 • 索引で重複値を許可する場合は、TRUE。 • 索引値を固有に規定する場合は、FALSE。 • TYPE 列でこの行が SQL_TABLE_STAT (その表自体の統計情報) に指示されている場合は、NULL が戻されます。
INDEX_QUALIFIER	VARCHAR(128)	索引名の修飾に使用する ID。 TYPE 列で SQL_TABLE_STAT が指示されている場合は、NULL になります。
INDEX_NAME	VARCHAR(128)	索引の名前。 TYPE 列の値が SQL_TABLE_STAT の場合、この列の値は NULL になります。
TYPE	NULL 以外の SMALLINT	結果セットのこの行に含める情報のタイプを指示します。 • SQL_TABLE_STAT この行に、この表自体の統計情報を含めるよう指示します。 • SQL_INDEX_CLUSTERED この行に索引の情報を含め、索引タイプをクラスター索引にするよう指示します。 • SQL_INDEX_HASHED この行に索引の情報を含め、索引タイプをハッシュ索引にするよう指示します。 • SQL_INDEX_OTHER この行に索引の情報を含め、索引タイプをクラスターまたはハッシュ以外の索引にするよう指示します。 **注:** 現時点で有効なのは、SQL_INDEX_OTHER だけです。
ORDINAL_POSITION	SMALLINT	INDEX_NAME 列で命名されている索引の列の序数桁位置。 TYPE 列の値が SQL_TABLE_STAT である場合、この列には NULL 値が戻されます。
COLUMN_NAME	VARCHAR(128)	索引の列の名前。
COLLATION	CHAR(1)	列のソート・シーケンス。昇順の場合は ″**A**″、降順の場合は ″**D**″。TYPE 列の値が SQL_TABLE_STAT の場合は、 NULL が戻されます。

SQLStatistics

表 170. SQLStatistics によって戻される列 (続き)

列名	データ・タイプ	説明
CARDINALITY	INTEGER	• TYPE 列の値が SQL_TABLE_STAT の場合、この列の値は表の行数になります。 • TYPE 列の値が SQL_TABLE_STAT の場合、この列の値は索引の中の固有値の数になります。 • DBMS の情報が使用可能でない場合は、NULL 値が戻されます。
PAGES	INTEGER	• TYPE 列の値が SQL_TABLE_STAT の場合、この列の値は表の保管に使用したページ数になります。 • TYPE 列の値が SQL_TABLE_STAT の場合、この列の値は索引の保管に使用したページ数になります。 • DBMS の情報が使用可能でない場合は、NULL 値が戻されます。

結果セットの行に表の統計が入っている (TYPE が SQL_TABLE_STAT に設定されている) 場合、NON_UNIQUE、INDEX_QUALIFIER、INDEX_NAME、 ORDINAL_POSITION、 COLUMN_NAME、および COLLATION の列の値は NULL に設定されます。 CARDINALITY または PAGES 情報が判別できない場合、これらの列には NULL が戻されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 171. SQLStatistics SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルに関する情報を要求しましたが、オープンされたカーソルはありません。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数またはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	3 つの部分で構成される表明のカタログ (先頭) 部分が、データ・ソースでサポートされていません。

SQLTablePrivileges - 表に関連した特権の入手

目的

SQLTablePrivileges() は、表と各表に関連した特権のリストを戻します。情報は SQL 結果セットに戻されますが、これは、照会で生成された結果セットの処理に使用するのと同じ関数を使って検索することができます。

構文

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

関数引き数

表 172. SQLTablePrivileges の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*szTableQualifier*	入力	3 分割の表名のカタログ修飾子。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	*cbTableQualifier*	入力	*CatalogName* の長さ。 0 に設定してください。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾する *pattern-value* を入れられるバッファー。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* の長さ。
SQLCHAR *	*TableName*	入力	表名で結果セットを修飾する *pattern-value* を保管する可能性のあるバッファー。
SQLSMALLINT	*NameLength3*	入力	*TableName* の長さ。

使用法

結果は、以下の表にリストされている列を含む標準結果セットとして戻されます。結果セットは、TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および PRIVILEGE ごとに配列されます。特定の表に複数の特権が関連付けられている場合は、各特権が別個の行として戻されます。

ここで報告される各特権の細分度は、列レベルで適用される場合とされない場合とがあります。たとえば、あるデータ・ソースの場合、表が更新できれば、その表の中の列もすべて更新できます。別のデータ・ソースの場合は、アプリケーションが SQLColumnPrivileges() を呼び出して、個々の列が同じ表特権を持っているかどうかを調べなければなりません。

多くの場合、SQLColumnPrivileges() の呼び出しは、システム・カタログに対する複雑な (そのため、経費のかさむ) 照会にマップされるので、慎重に使用する必要があり、何回も呼び出さなくて済むように結果を保管しておかなければなりません。

SQLTablePrivileges

カタログ関数結果セットの VARCHAR 列は、 SQL92 制限と一貫性があるように 128 という最大長属性で宣言されています。 DB2 名は 128 未満なので、アプリケーションは常に出力バッファー用に 128 文字 (およびヌル終止符) を取り分けておくか、あるいは、SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続されている DBMS がサポートしている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さをそれぞれ判別することができます。

今後のリリースでは、新しい列が追加されたり、既存の列名が変更されたりする可能性はありますが、現行列の位置は変更されません。

表 173. SQLTablePrivileges から戻される列

列名	データ・タイプ	説明
TABLE_CAT	VARCHAR(128)	これは常に NULL です。
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME が入っているスキーマの名前。
TABLE_NAME	NULL 以外の VARCHAR(128)	表の名前。
GRANTOR	VARCHAR(128)	特権を付与したユーザーの許可 ID。
GRANTEE	VARCHAR(128)	特権が付与されるユーザーの許可 ID。
PRIVILEGE	VARCHAR(128)	表の特権。これは以下のいずれかのストリングになります。 • ALTER • CONTROL • INDEX • DELETE • INSERT • REFERENCES • SELECT • UPDATE
IS_GRANTABLE	VARCHAR(3)	被認可者が他のユーザーに特権を付与することが許可されているかどうかを示します。 これは、YES、NO、または NULL になります。

注: DB2 CLI で使われる列名は、X/Open CLI CAE 仕様スタイルに準拠します。列のタイプ、内容、および順序は、 ODBC において SQLProcedures() の結果セット用に定義されているものと同じです。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 174. SQLTablePrivileges SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	ストリングまたはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、SQL_NTS と等価ではありませんでした。
HY010	関数シーケンス・エラー	ステートメント・ハンドルのカーソルがオープンしています。 このステートメント・ハンドル用の接続がありません。

制約事項

なし。

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);
```

参照

SQLTables - 表情報の取得

目的

SQLTables() は、接続されたデータ・ソースのシステム・カタログに保管されている表の名前と関連情報のリストを戻します。表名のリストは、結果セットとして戻されますが、このセットは SELECT ステートメントで生成された結果セットの検索に使用した関数と同じ関数で検索できるようになっています。

構文

```
SQLRETURN SQLTables  (SQLHSTMT       hstmt,
                      SQLCHAR        *szCatalogName,
                      SQLSMALLINT    cbCatalogName,
                      SQLCHAR        *szSchemaName,
                      SQLSMALLINT    cbSchemaName,
                      SQLCHAR        *szTableName,
                      SQLSMALLINT    cbTableName,
                      SQLCHAR        *szTableType,
                      SQLSMALLINT    cbTableType);
```

関数引き数

表 175. SQLTables の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*hstmt*	入力	ステートメント・ハンドル
SQLCHAR *	*szCatalogName*	入力	結果セットを修飾する *pattern-value* を含むバッファー。 *Catalog* は、 3 つの部分で構成される表名の最初の部分です。 NULL ポインターまたはゼロ長のストリングでなければなりません。
SQLSMALLINT	*cbCatalogName*	入力	*szCatalogName* の長さ。 0 に設定してください。
SQLCHAR *	*szSchemaName*	入力	スキーマ名で結果セットを修飾する *pattern-value* を含むバッファー。
SQLSMALLINT	*cbSchemaName*	入力	*szSchemaName* の長さ。
SQLCHAR *	*szTableName*	入力	表名で結果セットを修飾する *pattern-value* を含むバッファー。
SQLSMALLINT	*cbTableName*	入力	*szTableName* の長さ。

表 175. SQLTables の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*szTableType*	入力	表名で結果セットを修飾する *value list* を含むバッファー。 この値のリストは、該当するタイプの値をコンマで区切ったリストです。有効な表タイプ ID には、ALL、BASE TABLE、TABLE、VIEW、SYSTEM TABLE があります。*szTableType* 引き数が NULL ポインターまたはゼロの長さのストリングである場合、この値はこれらの表タイプ ID の有効値をすべて指定したときと等価になります。 SYSTEM TABLE が指定されると、システム表とシステム・ビュー (存在する場合) が両方とも戻されます。 表タイプを指定する際には引用符を使用してもしなくてもかまいません。
SQLSMALLINT	*cbTableType*	入力	*szTableType* のサイズ。

szCatalogName、*szSchemaName*、および *szTableName* の各引き数では、検索パターンが受け入れられることに注意してください。

ワイルドカード文字と一緒にエスケープ文字を指定して、検索パターン内で実際の文字が使われるようにすることができます。エスケープ文字は、SQL_ATTR_ESCAPE_CHAR 環境属性上に指定します。

使用法

表の情報は結果セットで戻されますが、この場合、それぞれの表は結果セットの 1 行で表されます。

SQLTables() によって戻される結果セットには、以下の表にリストする列がリスト順序で入れられます。

表 176. SQLTables によって戻される列

列名	データ・タイプ	説明
TABLE_CAT	VARCHAR(128)	現行サーバー。
TABLE_SCHEM	VARCHAR(128)	TABLE_NAME が入っているスキーマの名前。
TABLE_NAME	VARCHAR(128)	表、またはビュー、または別名、またはシノニムの名前。
TABLE_TYPE	VARCHAR(128)	TABLE_NAME 列の名前で指定されているタイプを識別します。ストリング値は、'TABLE'、'VIEW'、'BASE TABLE'、または 'SYSTEM TABLE' が有効です。
REMARKS	VARCHAR(254)	表に関する記述情報が入ります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

SQLTables

- SQL_INVALID_HANDLE

診断

表 177. SQLTables SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効	カーソルに関する情報を要求しましたが、オープンされたカーソルはありません。
40003 *	ステートメントの完了が不明	関数の処理完了前に、 CLI とデータ・ソースの間の通信リンクに障害が起こりました。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY009	引き数またはバッファー長が無効	名前長引き数のうち 1 つの値は 0 未満でしたが、SQL_NTS と等価ではありませんでした。
HYC00	ドライバーでサポートされていない	3 つの部分で構成される表明のカタログ (先頭) 部分が、データ・ソースでサポートされていません。

SQLTransact - トランザクション管理

目的

SQLTransact() は、接続中の現在のトランザクションをコミットまたはロールバックします。

接続時間内または SQLTransact() への直前 (最新) の呼び出し以来、この接続で実行されたすべてのデータベース変更がコミットまたはロールバックされます。

トランザクションが接続上で活動状態になっている場合は、アプリケーション・プログラムとデータベースとの間の切断処理に入る前にアプリケーション・プログラムから SQLTransact() を呼び出す必要があります。

構文

```
SQLRETURN SQLTransact (SQLHENV        henv,
                       SQLHDBC        hdbc,
                       SQLSMALLINT    fType);
```

関数引き数

表 178. SQLTransact の引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*henv*	入力	環境ハンドル。 *hdbc* が有効な接続ハンドルである場合、*henv* は無視されます。
SQLHDBC	*hdbc*	入力	データベース接続ハンドル *hdbc* が SQL_NULL_HDBC に設定されている場合、 *henv* の値は、接続に関連した環境ハンドルになっている必要があります。
SQLSMALLINT	*fType*	入力	トランザクションに対してとりたいアクション。この引き数の値は、以下のいずれかである必要があります。 • SQL_COMMIT • SQL_ROLLBACK • SQL_COMMIT_HOLD • SQL_ROLLBACK_HOLD

使用法

SQL_COMMIT または SQL_ROLLBACK でトランザクションを完了すると、次のような結果を生じます。

- SQLTransact() の呼び出しの後もステートメント・ハンドルは有効のままになります。
- カーソル名、バインド・パラメーター、および列バインドは、トランザクション完了後も有効のままになります。
- オープン・カーソルはクローズされ、検索保留になっている結果セットはすべて廃棄されます。

SQL_COMMIT_HOLD または SQL_ROLLBACK_HOLD でトランザクションを完了しても、データベースの変更はやはりコミットまたはロールバックされますが、カーソルがクローズされることはありません。

SQLTransact

接続上に現在活動状態のトランザクションが存在しない場合は、 SQLTransact() を呼び出してもデータベース・サーバーへの効果はなく、SQL_SUCCESS が戻されます。

COMMIT または ROLLBACK 中は、接続がないため、 SQLTransact() を実行しても失敗します。この場合、 COMMIT または ROLLBACK が処理されているかどうかはアプリケーション・プログラムには分からないので、データベース管理者に問い合わせる必要があるかもしれません。トランザクション・ログとその他のトランザクション管理作業の詳細については、 DBMS 製品情報を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 179. SQLTransact SQLSTATE

SQLSTATE	説明	解説
08003	接続がオープンしていない	*hdbc* は接続状態になっていません。
08007	トランザクション時に接続障害が発生	この関数の実行時に *hdbc* 関連の接続が失敗し、この障害の発生前に要求された COMMIT または ROLLBACK が実行されたかどうかが判別できません。
58004	システム・エラー	リカバリー不能なシステム・エラーです。
HY001	メモリーの割り振りの失敗	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーを割り振ることができません。
HY012	トランザクションの操作状態が無効	引き数 *fType* に指定された値が、 SQL_COMMIT にも SQL_ROLLBACK にもなっていませんでした。
HY013 *	メモリー管理の問題	ドライバーは、関数の実行または完了をサポートするのに必要なメモリーにアクセスできませんでした。

例

109 ページの『例』を参照してください。

付録 A. DB2 UDB CLI 一般診断情報

この付録のセクションには、この資料の中の各セクションで言及された情報が表にして示してあります。

DB2 UDB CLI 関数戻りコード

戻りコード	値	説明
SQL_SUCCESS	0	関数は正常に完了し、追加の SQLSTATE 情報はありません。
SQL_SUCCESS_WITH_INFO	1	関数は正常に完了しましたが、警告またはその他の情報があります。 SQLSTATE およびその他のエラー情報を受け取るには、 SQLError() を呼び出してください。
SQL_NO_DATA_FOUND	100	関数は正常に戻りましたが、関連情報が見つかりません。
SQL_ERROR	-1	関数は失敗しました。 SQLSTATE およびその他のエラー情報を受け取るには、 SQLError() を呼び出してください。
SQL_INVALID_HANDLE	-2	関数は、無効なハンドル (環境、接続、またはステートメント・ハンドル) が入力引き数として渡されたために失敗しました。

付録 B. DB2 UDB CLI のインクルード・ファイル

DB2 UDB CLI で使用されるインクルード・ファイルは sqlcli.h のみです。

```
/*** 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 */
```

付録 C. DB2 UDB CLI のアプリケーション・コード・リストの例

この付録では、本書全体を通して使われた例の完全なコードのリストを示しています。

詳細なエラー・チェックはサンプルの中に組み込まれていません。

以下を参照してください。

- 『例: 組み込み SQL とそれと同等の DB2 UDB CLI 関数呼び出し』
- 296 ページの『例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し』

例: 組み込み SQL とそれと同等の DB2 UDB CLI 関数呼び出し

この例では、組み込みステートメントが注釈で示されており、次いで DB2 UDB CLI 関数呼び出しが示されています。

コード例については、 viii ページの『コードの特記事項情報』を参照してください。

```
/************************************************************************
** 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);

}
```

例: 対話式 SQL とそれと同等の DB2 UDB CLI 関数呼び出し

この例は、対話式 SQL ステートメントの実行を示しており、 7 ページの『第 2 章 DB2 UDB CLI アプリケーションの作成』に記述されている流れに従っています。

コード例については、 viii ページの『コードの特記事項情報』を参照してください。

```
/*****************************************************************************
** 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);

}
```

付録 D. サーバー・モードでの DB2 UDB CLI の実行

『DB2 UDB CLI を SQL サーバー・モードで実行する理由』を参照してください。

DB2 UDB CLI を SQL サーバー・モードで実行する理由

SQL サーバー・モードで実行するのは、多数のアプリケーション・プログラムが、データベース・サーバーとして働く必要があるからです。つまり、1 つのジョブが、複数のユーザーのために SQL 要求を実行することを意味します。アプリケーション・プログラムは、SQL サーバー・モードを使用しないと、次に示す 3 つの制限事項のいずれかによる規制を受けることがあります。

1. 1 つのジョブは、活動化グループごとに 1 つのコミット・トランザクションしかもつことができない。
2. 1 つのジョブは、一度に 1 つの RDB にしか接続できない。
3. 接続上で渡されるユーザー ID に関係なく、すべての SQL ステートメントは、ジョブのユーザー・プロファイルのもとで実行される。

SQL サーバー・モードを使用すると、すべての SQL ステートメントが別々のジョブに経路指定されるので、上記のような制限事項を免れることができます。各接続は、それぞれ独自のジョブにおいて実行されます。システムは、QSYSWRK サブシステム内の事前開始ジョブを使って、各接続の起動時間を最短化します。 SQLConnect の呼び出しごとにそれぞれ異なるユーザー・プロファイルの受諾が可能なので、おのおののジョブも、それぞれ独自のコミット・トランザクションをもつことができます。 SQLDisconnect の実行が完了すると、ジョブはリセットされ、使用可能なジョブのプールに書き戻されます。

DB2 UDB CLI を SQL サーバー・モードで実行することの詳細については、以下を参照してください。

- 『SQL サーバー・モードでの DB2 UDB CLI の始動』
- 304 ページの『サーバー・モードでの DB2 UDB CLI の実行の制約事項』

SQL サーバー・モードでの DB2 UDB CLI の始動

ジョブを SQL サーバー・モードにするには、次の 2 通りの方法があります。

1. CLI 関数 SQLSetEnvAttr を使用するのが、最も可能性の高いケースです。 SQL サーバー・モードは、CLI アプリケーション・プログラムに最も適しています。このアプリケーション・プログラムでは、複数の接続ハンドルの概念がすでに用いられているからです。このモードは、CLI 環境を割り振った直後に設定します。さらに、このモードの設定の前に、ジョブが、いずれかの SQL を実行していたり、コミットメント制御を開始していたりしてはなりません。この 2 つのケースのどちらかに該当する場合、モードはサーバー・モードに変更されないで、 SQL は引き続き「インライン」で実行されます。

   例

```
.
SQLAllocEnv(&henv);
long  attr;
attr  =  SQL_TRUE
SQLSetEnvAttr(henv,SQL_ATTR_SERVER_MODE,&attr,0);
SQLAllocConnect(henv,&hdbc);
.
.
```

2. サーバー・モードを設定する 2 番目の方法では、ジョブの変更 (QWTCHGJB) API を使います。 QWTCHGJB API の詳細は、iSeries Information Center 中のトピック『OS/400 API』を参照してください。

SQL サーバー・モードの設定が完了したら、すべての SQL 接続と SQL ステートメントはサーバー・モードで稼働します。逆方向または順方向への切り替えはありません。ジョブは、いったんサーバー・モードに入ったら、コミットメント制御を開始できなくなるので、対話式 SQL も使えなくなります。

サーバー・モードでの DB2 UDB CLI の実行の制約事項

- ジョブでは、処理の開始で他の何よりも先に、サーバー・モードを設定しなければなりません。 CLI を中心的に使用するジョブでサーバー・モードにするには、 SQLSetEnvAttr 呼び出しを使用しなければなりません。これは、SQLAllocEnv の直後に、他の何よりも先に行うことに注意してください。いったんサーバー・モードをオンにすると、オフにできなくなります。
- すべての SQL 関数は、事前開始ジョブとコミットメント制御で実行されます。サーバー・モードに入る前も後も、起動側のジョブでコミットメント制御を始動しないでください。
- SQL は事前開始ジョブ内で処理されるので、起動側のジョブにおける特定の変更に対して注意は払われません。それには、ライブラリー・リスト、ジョブ優先順位、メッセージ・ログ、などの変更も含まれます。事前開始では、起動側のジョブにおける CCSID 値の変更は重視されます。これは、元のユーザー・プログラムにデータがマップされる方法に影響を与えるからです。
- アプリケーション・プログラムは、サーバー・モードで実行するときには、組み込まれているかまたは SQL CLI による SQL コミットとロールバックを使用する必要があります。 CL コマンドは使えません。起動側のジョブにおいてコミットメント制御は実行されないからです。ジョブは、接続の切断の前に COMMIT を出さなければなりません。そうしないと、暗黙のロールバックが行われます。
- サーバー・モードのジョブから対話式 SQL を使用することはできません。サーバー・モードのときに STRSQL を使うと、 SQL6141 メッセージが出されることになります。
- しかも、サーバー・モードでは、SQL コンパイルも実行できません。コンパイル済みの SQL プログラムを実行するのにサーバー・モードを使用することはできますが、コンパイルの場合にこのモードをオンにしてはなりません。ジョブがサーバー・モードになっていると、コンパイルは失敗します。
- SQLDataSources は、実行するのに接続ハンドルを必要としないという点で特異です。プログラムは、サーバー・モードで SQLDataSources を使用するには、あらかじめローカル・データベースに接続しておかなければなりません。 DataSources は接続先の RDB の名前を見つけ出すのに使われるので、 IBM は SQLConnect 上での RDB 名用の NULL ポインターの引き渡しをサポートしています。それによって、ローカル接続が確立されます。このようにして、事前にシステム名が分かっていない場合に、総称プログラムを作成できるようになっています。
- CLI を介してコミットおよびロールバックを行うときは、 SQLEndTran と SQLTransact の呼び出しに接続ハンドルを含める必要があります。サーバー・モードで実行しないときは、すべてをコミットするための接続ハンドルを省略してもかまいません。ただし、それはサーバー・モードではサポートされません。接続 (またはスレッド) ごとに、それぞれ独自のトランザクションのスコープ化があるからです。
- SQL サーバー・モードでの実行で、複数のスレッドに接続ハンドルを共用させることはお勧めしません。これは、いずれかのスレッドがこれから処理しようとしている戻りデータやエラー情報を、他のスレッドが上書きすることがあるためです。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アプリケーション
  サンプル 293
  タスク 7
  例 293
インクルード・ファイル 275
エラー情報、検索 98, 100
大文字小文字の区別 21

[カ行]

カーソル 4, 15
カーソル名の取得、関数 139, 142
カーソル名の設定、関数 240, 241
外部キー列、関数 116
外部キー列名、関数 120
解放
  環境の解放、関数 230
  環境ハンドル、関数 123, 124, 125, 229
  ステートメント・ハンドル、関数 127, 129
  接続ハンドル、関数 121, 122
  ハンドル、関数 126
拡張取り出し、関数 105
可搬性 6
環境属性の取得、関数 155, 156
環境属性の設定、関数 246, 250
環境の解放
  ReleaseEnv、関数 230
環境ハンドル 4
  解放 8
  解放、関数 123, 124, 125, 229
  割り振り 8
  割り振り、関数 30
関数の取得、関数 156, 158
記述子フィールドの設定、関数 242, 243, 244, 245
記述子レコードの取得、関数 147, 148
記述フィールドの取得、関数 144, 146
基本キー列、関数 213, 215
基本表の索引情報と統計情報の取得、関数 262, 264
行数の取得、関数 231, 232
切り捨て 21
組み込み SQL 293
結果列の数、関数 203, 204
結果列の数の取得 203
言語情報、関数 192
言語または準拠情報の取得、関数 193
コア・レベル関数 1
コミット 15
固有の SQL テキスト、関数 196, 198

[サ行]

サーバー・モード
  開始 303
  制約事項 304
作成 7
作成、ステートメントの 12
サンプル・アプリケーション 293
実行、ステートメント 12
実行ステートメント、関数 103, 104
終了 7, 8
紹介、CLI の 1
情報の取得、関数 159, 173
初期設定 7, 8
診断 16
診断情報を戻す 149, 152
診断フィールド情報を戻す 151
診断レコード情報を戻す 154
据え置き引き数 13
ステートメント属性の取得、関数 179, 181
ステートメント属性の設定、関数 252, 255
ステートメントの準備作成、関数 209, 212
ステートメントの直接実行、関数 101, 102
ステートメントの取り消し、関数 61
ステートメント・オプションの取得、関数 182, 183
ステートメント・オプションの設定、関数 256, 257
ステートメント・ハンドル 4
  解放 15
  解放、関数 127, 129
  最大数 12
  割り振り 12
  割り振り、関数 35
ストリング値の一部の検索、関数 184
ストリング値の長さの検索、関数 174
ストリングの開始位置を戻す、関数 176
ストリング引き数 20, 21
制限付きハンドル、定義 30
静的 SQL 6
接続、関数 75, 77
接続オプションの取得、関数 138
接続オプションの設定、関数 238, 239
接続属性の取得、関数 136, 137
接続属性の設定、関数 233, 237

接続ハンドル 4
解放 9
解放、関数 121, 122
割り振り 9
割り振り、関数 27
切断、関数 90, 91

[タ行]

タイプ情報の入手、関数 187
他の結果セット、関数 194, 195
直接実行 12
次の結果セット、関数 199, 200
データの取得、関数 143
データ変換
説明 19
データ・タイプ 17
デフォルト・データ・タイプ 17
C データ・タイプ 17
SQL データ・タイプ 17
データ・ソースの入手、関数 79, 82
データ・タイプ
総称 18
C 17, 18
ODBC 18
SQL 17
定義
制限付きハンドル 30
動的 SQL 6
特殊な列 (行 ID) の取得、関数 261
特殊列名の取得、関数 258
トランザクション管理 15
トランザクション管理、関数 271
トランザクション管理の終わり、関数 96
トランザクション処理 7
取り出し、関数 108, 113

[ハ行]

バインド
パラメーター・マーカー 13
列 14
パラメーター数、関数 201, 202
パラメーターの設定、関数 251
パラメーターのデータを設定する、関数 226, 228
パラメーター・オプション、関数 207
パラメーター・データ、関数 205, 206
パラメーター・マーカー 4
パラメーター・マーカー、バインド 13
パラメーター・マーカーに対するバッファーのバインド、関数 48, 52, 53, 61
ハンドル
解放、関数 126
環境ハンドル 4, 8
ステートメント・ハンドル 4
接続ハンドル 4, 8
表情報の取得、関数 268, 270
表に関連した特権の入手 265, 267
表の列に関連した特権の入手、関数 68
表の列名の入手、関数 70, 74
ファイル参照のバインド、関数 42
ファイル参照の割り当て、関数 45
プロシージャーのパラメーターの入手、関数 221
プロシージャー名リストの入手 222
プロシージャー名リストの入手、関数 225
プロシージャー・パラメーター情報、関数 216
ヘッダー・ファイル 275

[マ行]

文字ストリング 20, 21
戻りコード 16, 273

[ラ行]

例、アプリケーション 293
列情報、関数 71
列属性、関数 63, 67, 267
列属性の記述、関数 83, 86
列特権、関数 52
列のバインド、関数 37, 41
ロールバック 15

[ワ行]

割り振り
環境ハンドル、関数 30, 32
ステートメント・ハンドル、関数 35, 36
接続ハンドル、関数 27, 29
割り振られたハンドル、関数 34
割り振りハンドル、関数 33
割り振りハンドル
割り振り、関数 33

B

BindFileToParam、関数 47

C

CLI
DB2 UDB CLI アプリケーションの作成 7

CLI 関数
SQLSetEnvAttr 303
CloseCursor ステートメント、関数 62
ColumnPrivileges、関数 70
CopyDesc ステートメント、関数 78

D

DriverConnect、関数 92, 95

F

FetchScroll、関数 114, 115

G

Get Col、関数 135
GetCol、関数 130

I

INVALID_HANDLE 16
ISO 標準 9075-3:1999 1

N

NULL 文字で終了するストリング 20

O

ODBC
および DB2 UDB CLI 1
カーソル名 140
コア・レベル関数 1
精度 72
SQLSTATE 17

S

SELECT 13
SQL
ステートメント
DELETE 15
SELECT 13
UPDATE 15
ステートメントの作成と実行 12
静的 6
動的 6
動的に作成される 4
パラメーター・マーカー 13
SQL サーバー・モード 303
SQLAllocConnect、関数
概説 8
説明 27, 29
SQLAllocEnv、関数
概説 8
説明 30, 32, 34
SQLAllocHandle、関数
説明 33
SQLAllocStmt、関数
概説 10
説明 35, 36
SQLBindCol、関数
概説 10, 14
説明 37, 41
SQLBindFileToCol、関数
説明 42
SQLBindFileToParam、関数
説明 45, 47
SQLBindParameter、関数
説明 53, 61
SQLBindParam、関数
説明 48, 52
SQLCancel、関数
説明 61
SQLCloseCursor、関数
説明 62
SQLColAttributes、関数
概説 10, 14
説明 63, 67, 267
SQLColumnPrivileges、関数
説明 52, 68, 70
SQLColumns、関数
説明 70, 71, 74
SQLConnect、関数
概説 8
説明 75, 77
SQLCopyDesc、関数
説明 78
SQLDataSources、関数
概説 10, 14
説明 79, 82
SQLDescribeCol、関数
概説 10, 14
説明 83, 86
SQLDescribeParam、関数
説明 87
SQLDisconnect、関数
概説 8
説明 90, 91
SQLDriverConnect、関数
説明 92, 95

SQLEndTranm、関数
説明 96
SQLError、関数
説明 98, 100
SQLExecDirect、関数
概説 10, 12
説明 101, 102
SQLExecute、関数
概説 10, 12
説明 103, 104
SQLExtendedFetch、関数
説明 105
SQLFetchScroll、関数
説明 114, 115
SQLFetch、関数
概説 10, 14
説明 108, 113
SQLForeignKeys、関数
説明 116, 120
SQLFreeConnect、関数
概説 8
説明 121, 122
SQLFreeEnv、関数
概説 8
説明 123, 124
SQLFreeHandle、関数
説明 125, 126
SQLFreeStmt、関数
概説 10
説明 127, 129
SQLGetCol、関数
説明 130, 135
SQLGetConnectAttr、関数
説明 136, 137
SQLGetConnectOption、関数
説明 138
SQLGetCursorName、関数
説明 139, 142
SQLGetData、関数
概説 10, 14
説明 143
SQLGetDescField、関数
説明 144, 146
SQLGetDescRec、関数
説明 147, 148
SQLGetDiagField、関数
説明 149, 151
SQLGetDiagRec、関数
説明 152, 154
SQLGetEnvAttr、関数
説明 155, 156
SQLGetFunctions、関数
説明 156, 158
SQLGetInfo、関数
説明 159, 173
SQLGetLength、関数
説明 174
SQLGetPosition、関数
説明 176
SQLGetStmtAttr、関数
説明 179, 181
SQLGetStmtOption、関数
説明 182, 183
SQLGetSubString、関数
説明 184
SQLGetTypeInfo、関数
説明 187, 191
SQLLanguages、関数
説明 192, 193
SQLMoreResults、関数
説明 194, 195
SQLNativeSql、関数
説明 196, 198
SQLNextResult、関数
説明 199, 200
SQLNumParams、関数
説明 201, 202
SQLNumResultCols、関数
概説 10, 14
説明 203, 204
SQLParamData、関数
説明 205, 206
SQLParamOptions、関数
説明 207
SQLPrepare、関数
概説 10, 12, 14
説明 209, 212
SQLPrimaryKeys、関数
説明 213, 215
SQLProcedureColumns、関数
説明 216, 221
SQLProcedures、関数
説明 222, 225
SQLPutData、関数
説明 226, 228
SQLReleaseEnv、関数
説明 229, 230
SQLRowCount、関数
概説 10
説明 231, 232
SQLSetConnectAttrm、関数
説明 233, 237

SQLSetConnectOption、関数
　説明　238, 239
SQLSetCursorName、関数
　説明　240, 241
SQLSetDescField、関数
　説明　242, 243
SQLSetDescRec、関数
　説明　244, 245
SQLSetEnvAttr、関数
　説明　246, 250
SQLSetParam、関数
　概説　10, 12, 13, 14
　説明　251
SQLSetStmtAttr、関数
　説明　252, 255
SQLSetStmtOption、関数
　説明　256, 257
SQLSpecialColumns、関数
　説明　258, 261
SQLSTATE　4, 17
SQLSTATE のフォーマット　17
SQLStatistics、関数
　説明　262, 264
SQLTablePrivileges、関数
　説明　265, 267
SQLTables、関数
　説明　268, 270
SQLTransact、関数
　概説　10, 14, 15
　説明　271
SQL_ERROR　16
SQL_NO_DATA_FOUND　16
SQL_NTS　20
SQL_SUCCESS　16
SQL_SUCCESS_WITH_INFO　16

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12