IBM DB2 10.1
for Linux, UNIX, and Windows

コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻

SA88-4677-00
(英文原典：SC27-3867-00)

IBM DB2 10.1
for Linux, UNIX, and Windows

コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻

SA88-4677-00
(英文原典：SC27-3867-00)

ご注意

本書および本書で紹介する製品をご使用になる前に、619 ページの『付録 B. 特記事項』に記載されている情報をお読みください。

原典： SC27-3867-00
IBM DB2 10.1
for Linux, UNIX, and Windows
Call Level Interface Guide and
Reference Volume 2

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2012.4

目次

本書について

「*コール・レベル・インターフェース (CLI) ガイドおよびリファレンス*」は、次の 2 巻に分かれています。

- 第 1 巻では、CLI を使用して DB2® Database for Linux, UNIX, and Windows 用のデータベース・アプリケーションを作成する方法を説明します。
- 第 2 巻は、CLI の関数、キーワード、および構成を説明するリファレンスです。

第 1 章 CLI と ODBC 関数のサマリー

ODBC 列の **Depr** は、ODBC に使用すべきでない関数を示しています。

SQL/CLI 列には、以下の値が入ります。

95 この関数は SQL/CLI 9075-3 仕様に定義されています。

SQL3 この関数は SQL/CLI 9075-3 に関する ISO SQL3 修正草案の SQL/CLI 部分に定義されています。

表 1. カテゴリー別の CLI 関数リスト

タスク 関数名	ODBC 3.0	SQL/CLI	DB2 CLI 最初のバージョンサポートされる	目的
データ・ソースへの接続				
SQLConnect()	Depr	95	V1.1	接続ハンドルを獲得します。
SQLAllocEnv()	Depr	95	V1.1	環境ハンドルを獲得します。 1 つ以上の接続に 1 つの環境ハンドルが使用されます。
SQLAllocHandle()	コア	95	V5	ハンドルを獲得します。
SQLBrowseConnect()	レベル 1	95	V5	データ・ソースに接続するのに必要な属性を獲得します。
SQLConnect()	コア	95	V1.1	データ・ソース名、ユーザー ID、およびパスワードを使用して特定のドライバーに接続します。
SQLDriverConnect()	コア	SQL3	V2.1 [1]	接続ストリングを使用して特定のドライバーに接続するか、またはオプションでドライバー・マネージャーとドライバーがユーザー用の接続ダイアログを表示するよう要求します。 **注:** この関数は、ODBC.INI ファイルでサポートされる追加の IBM キーワードの影響も受けます。
SQLDrivers()	コア	不可	なし	CLIは、この関数がドライバー・マネージャーによってインプリメントされているため、この関数をサポートしません。
SQLSetConnectAttr()	コア	95	V5	接続属性を設定します。
SQLSetConnectOption()	Depr	95	V2.1	接続属性を設定します。
SQLSetConnection()	不可	SQL3	V2.1	現行のアクティブな接続を設定します。この関数は、複数の同時接続がある CLI アプリケーションで組み込み SQL を使用するときだけ使用する必要があります。
ドライバーおよびデータ・ソースに関する情報の獲得				
SQLDataSources()	Lvl 2	95	V1.1	使用できるデータ・ソースのリストを返します。
SQLGetInfo()	コア	95	V1.1	特定のドライバーおよびデータ・ソースに関する情報を返します。
SQLGetFunctions()	コア	95	V1.1	サポートされているドライバー関数のリストを返します。
SQLGetTypeInfo()	コア	95	V1.1	サポートされているデータ・タイプに関する情報を返します。
ドライバー・オプションの設定および取り出し				

表 1. カテゴリー別の CLI 関数リスト (続き)

タスク 関数名	ODBC 3.0	SQL/CLI	DB2 CLI 最初のバージョンサポートされる	目的
SQLCreatePkg()	不可	不可	V9.5	パッケージをデータベースにバインドします。
SQLSetEnvAttr()	コア	95	V2.1	環境オプションを設定します。
SQLGetEnvAttr()	コア	95	V2.1	環境オプションの値を返します。
SQLGetConnectAttr()	Lvl 1	95	V5	接続オプションの値を返します。
SQLGetConnectOption()	Depr	95	V2.1 [1]	接続オプションの値を返します。
SQLSetStmtAttr()	コア	95	V5	ステートメント属性を設定します。
SQLSetStmtOption()	Depr	95	V2.1 [1]	ステートメント・オプションを設定します。
SQLGetStmtAttr()	コア	95	V5	ステートメント属性の値を返します。
SQLGetStmtOption()	Depr	95	V2.1 [1]	ステートメント・オプションの値を返します。
SQLReloadConfig()	不可	不可	V9.7	クライアント構成ファイル db2dsdriver.cfg から構成プロパティーを再ロードします。
SQL 要求の準備				
SQLAllocStmt()	Depr	95	V1.1	ステートメント・ハンドルを割り振ります。
SQLPrepare()	コア	95	V1.1	後で実行するための SQL ステートメントを準備します。
SQLExtendedPrepare()	不可	不可	V6	その後の実行のために SQL ステートメントのステートメント属性の配列を準備します。
SQLExtendedBind()	不可	不可	V6	SQLBindCol() および SQLBindParameter() を繰り返し呼び出す代わりに、列の配列をバインドします。
SQLBindParameter()	Lvl 1	95 [2]	V2.1	SQL ステートメントのパラメーター用のストレージを割り当てます (ODBC 2.0)。
SQLSetParam()	Depr	不可	V1.1	SQL ステートメントのパラメーター用のストレージを割り当てます (ODBC 1.0)。 **注:** ODBC 2.0 では、この関数の代わりに SQLBindParameter() が用いられます。
SQLParamOptions()	Depr	不可	V2.1	パラメーター用の複数の値の使用を指定します。
SQLGetCursorName()	コア	95	V1.1	ステートメント・ハンドルに関連したカーソル名を返します。
SQLSetCursorName()	コア	95	V1.1	カーソル名を指定します。
要求のサブミット				
SQLDescribeParam()	レベル 2	SQL3	V5	パラメーター・マーカーの記述を返します。
SQLExecute()	コア	95	V1.1	準備済みステートメントを実行します。
SQLExecDirect()	コア	95	V1.1	ステートメントを実行します。
SQLNativeSql()	Lvl 2	95	V2.1 [1]	ドライバーによって変換された SQL ステートメントのテキストを返します。
SQLNumParams()	Lvl 2	95	V2.1 [1]	ステートメント内のパラメーターの数を返します。
SQLParamData()	Lvl 1	95	V2.1 [1]	SQLPutData() と組み合わせて使用され、実行時にパラメーター・データを渡します。これは長いデータ値の場合に便利です。
SQLPutData()	コア	95	V2.1 [1]	パラメーターのデータ値の一部または全部を送ります。これは長いデータ値の場合に便利です。
結果および結果に関する情報の取り出し				

表 1. カテゴリー別の CLI 関数リスト (続き)

タスク 関数名	ODBC 3.0	SQL/CLI	DB2 CLI 最初の バージョン サポート される	目的
SQLRowCount()	コア	95	V1.1	挿入、更新、または削除の要求によって影響を受ける行の数を返します。
SQLNumResultCols()	コア	95	V1.1	結果セット内の列の数を返します。
SQLDescribeCol()	コア	95	V1.1	結果セット内の列について記述します。
SQLColAttribute()	コア	可	V5	結果セット内の列の属性を記述します。
SQLColAttributes()	Depr	可	V1.1	結果セット内の列の属性を記述します。
SQLColumnPrivileges()	レベル 2	95	V2.1	表の列に関連した特権を獲得します。
SQLSetColAttributes()	不可	不可	V2.1	結果セット内の列の属性を設定します。
SQLBindCol()	コア	95	V1.1	結果列のためのストレージを割り当て、データ・タイプを指定します。
SQLFetch()	コア	95	V1.1	結果行を返します。
SQLFetchScroll()	コア	95	V5	結果行から行セットを返します。
SQLExtendedFetch()	Depr	95	V2.1	複数の結果行を返します。
SQLGetData()	コア	95	V1.1	結果セットの 1 行の 1 列の一部または全部を返します。これは長いデータ値の場合に便利です。
SQLMoreResults()	Lvl 1	SQL3	V2.1 [a]	使用できる結果セットがまだ残っているかどうかを判別し、もし残っていれば、次の結果セットの処理を初期化します。
SQLNextResult()	不可	可	V7.1	ストアード・プロシージャーから戻された複数の結果セットに順不同でアクセスすることができます。
SQLError()	Depr	95	V1.1	追加のエラー情報または状況情報を返します。
SQLGetDiagField()	コア	95	V5	診断データのフィールドを獲得します。
SQLGetDiagRec()	コア	95	V5	診断データの複数のフィールドを獲得します。
SQLSetPos()	レベル 1	SQL3	V5	行セット内のカーソル位置を設定します。
SQLGetSQLCA()	不可	不可	V2.1	ステートメント・ハンドルに関連した SQLCA を返します。
SQLBulkOperations()	レベル 1	不可	V6	ブックマークを使用して大量の追加、更新、削除、およびフェッチを実行します。
記述子				
SQLCopyDesc()	コア	95	V5	記述子情報をハンドル間でコピーします。
SQLGetDescField()	コア	95	V5	記述子レコードの単一のフィールド設定を獲得します。
SQLGetDescRec()	コア	95	V5	記述子レコードの複数のフィールド設定を獲得します。
SQLSetDescField()	コア	95	V5	記述子レコードの単一のフィールドを設定します。
SQLSetDescRec()	コア	95	V5	記述子レコードの複数のフィールド設定を設定します。
ラージ・オブジェクトのサポート				
SQLBindFileToCol()	不可	不可	V2.1	LOB ファイル参照を LOB 列に関連付けます。
SQLBindFileToParam()	不可	不可	V2.1	LOB ファイル参照をパラメーター・マーカーに関連付けます。

表 1. カテゴリー別の CLI 関数リスト (続き)

タスク 関数名	ODBC 3.0	SQL/CLI	DB2 CLI 最初の バージョン サポート される	目的
SQLGetLength()	不可	SQL3	V2.1	LOB ロケーターで参照されるストリングの長さを取得します。
SQLGetPosition()	不可	SQL3	V2.1	LOB ロケーターで参照されるソース・ストリング内のストリングの位置を取得します。
SQLGetSubString()	不可	SQL3	V2.1	ソース・ストリング内のサブストリングを参照する新しい LOB ロケーターを作成します。ソース・ストリングも LOB ロケーターで表されます。
データ・ソースのシステム表に関する情報の獲得 (カタログ関数)				
SQLColumns()	Lvl 1	SQL3	V2.1 [1]	指定した表の中の列名のリストを返します。
SQLExtendedProcedures()	不可	不可	V9.7	特定のデータ・ソースに保管されたプロシージャー名のリストを、追加情報と一緒に返します。
SQLExtendedProceduresColumns()	不可	不可	V9.7	指定したプロシージャーの入力パラメーターおよび出力パラメーターのリストを、追加情報と一緒に返します。
SQLForeignKeys()	Lvl 2	SQL3	V2.1	指定した表の外部キーがある場合には、外部キーを構成する列名のリストを返します。
SQLPrimaryKeys()	Lvl 1	SQL3	V2.1	表の主キーを構成する列名のリストを返します。
SQLProcedureColumns()	Lvl 2	不可	V2.1	指定したプロシージャーの入力パラメーターおよび出力パラメーターのリストを返します。
SQLProcedures()	Lvl 2	不可	V2.1	特定のデータ・ソースに保管されたプロシージャー名のリストを返します。
SQLSpecialColumns()	コア	SQL3	V2.1 [1]	指定した表の行を固有に識別する最適な列の集まりに関する情報を返します。
SQLStatistics()	コア	SQL3	V2.1 [1]	単一の表およびその表に関連した索引のリストに関する統計を返します。
SQLTablePrivileges()	Lvl 2	SQL3	V2.1	表のリストおよび各表に関連した特権のリストを返します。
SQLTables()	コア	SQL3	V2.1 [1]	特定のデータ・ソースに保管された表名のリストを返します。
ステートメントの終了				
SQLFreeHandle()	コア	95	V1.1	ハンドル・リソースを解放します。
SQLFreeStmt()	コア	95	V1.1	ステートメント処理を終了し、関連したカーソルをクローズし、ペンディング中の結果を廃棄し、オプションで、ステートメント・ハンドルに関連したすべてのリソースを解放します。
SQLCancel()	コア	95	V1.1	SQL ステートメントを取り消します。
SQLTransact()	Depr	不可	V1.1	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。
SQLCloseCursor()	コア	95	V5	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。
接続の終了				
SQLDisconnect()	コア	95	V1.1	接続をクローズします。
SQLEndTran()	コア	95	V5	接続のトランザクションを終了します。
SQLFreeConnect()	Depr	95	V1.1	接続ハンドルを解放します。

表 1. カテゴリー別の CLI 関数リスト (続き)

タスク 関数名	ODBC 3.0	SQL/CLI	DB2 CLI 最初の バージョン サポート される	目的
SQLFreeEnv()	Depr	95	V1.1	環境ハンドルを解放します。
データベースの作成とドロップ				
SQLCreateDb()	不可	不可	V9.7	指定のデータベース名、コード・セット、モードに基づいてデータベースを作成します。
SQLDropDb()	不可	不可	V9.7	指定のデータベースをドロップします。

注:

[1] この関数の実行時サポートは、DB2 Client Application Enabler for DOS バージョン 1.2 製品でも利用することができました。

[2] SQLBindParam() に代わって SQLBindParameter() が使用されています。

ODBC 関数には次の制限が適用されます。

- SQLSetScrollOptions() は、すでに SQL_CURSOR_TYPE、SQL_CONCURRENCY、SQL_KEYSET_SIZE、および SQL_ROWSET_SIZE ステートメント・オプションに置き換えられているため、実行時の使用でのみサポートされます。
- SQLDrivers() は、ODBC Driver Manager によってインプリメントされています。

Unicode 関数 (CLI)

CLI Unicode 関数は、ANSI ストリング引数の代わりに Unicode ストリング引数を受け入れます。 Unicode ストリング引数は、 UCS-2 エンコード (ネイティブ・エンディアン形式) でなければなりません。 ODBC API 関数には、それぞれのストリング引数の形式を示す接尾部があります。すなわち、 Unicode を受け入れる場合の接尾部は W であり、ANSI を受け入れる場合は接尾部がありません (ODBC では、名前の末尾が A の同等の関数が追加されていますが、これらは CLI では提供されません)。次に示すのは、ANSI バージョンと Unicode バージョンの両方で使用できる CLI 関数のリストです。

- SQLBrowseConnect
- SQLColAttribute
- SQLColAttributes
- SQLColumnPrivileges
- SQLColumns
- SQLConnect
- SQLCreateDb
- SQLDataSources
- SQLDescribeCol
- SQLDriverConnect
- SQLDropDb
- SQLError
- SQLExecDirect

- SQLExtendedPrepare
- SQLExtendedProcedures
- SQLExtendedProcedureColumns
- SQLForeignKeys
- SQLGetConnectAttr
- SQLGetConnectOption
- SQLGetCursorName
- SQLGetDescField
- SQLGetDescRec
- SQLGetDiagField
- SQLGetDiagRec
- SQLGetInfo
- SQLGetPosition
- SQLGetStmtAttr
- SQLNativeSQL
- SQLPrepare
- SQLPrimaryKeys
- SQLProcedureColumns
- SQLProcedures
- SQLReloadConfig
- SQLSetConnectAttr
- SQLSetConnectOption
- SQLSetCursorName
- SQLSetDescField
- SQLSetStmtAttr
- SQLSpecialColumns
- SQLStatistics
- SQLTablePrivileges
- SQLTables

常にストリングの長さである引数を持つ Unicode 関数は、それらの引数を、ストリングを格納するのに必要な SQLWCHAR エレメントの数として解釈します。サーバー・データに対して長さの情報を返す関数でも、表示サイズと精度は、それらを格納するための SQLWCHAR エレメントの数で示されます。長さ (データの転送サイズ) がストリングまたはストリング以外のデータを参照する場合、それはそのデータを格納するために必要なバイト数として解釈されます。

例えば、SQLGetInfoW() は長さをバイト数として解釈しますが、 SQLExecDirectW() は SQLWCHAR エレメント数を使用します。 UTF-16 拡張文字セットの 1 文字について考慮してみましょう (UTF-16 は UCS-2 の拡張文字セットの 1 つです。 Microsoft Windows 2000 および Microsoft Windows XP では UTF-16 が使用されています)。 Microsoft Windows 2000 では、その 1 文字を格納するために 2 個の SQL_C_WCHAR、したがって 4 バイトが使用されます。それで、この文字の表示サ

イズは 1、ストリング長は 2 (SQL_C_WCHAR を使用した場合)、そしてバイト・カウントは 4 になります。CLI は結果セットからのデータを、アプリケーションのバインドに応じて Unicode または ANSI で返します。アプリケーションが SQL_C_CHAR にバインドする場合、ドライバーは SQL_WCHAR データを SQL_CHAR に変換します。 ODBC Driver Manager は (使用されている場合)、ANSI ドライバーについては SQL_C_WCHAR を SQL_C_CHAR にマップしますが、 Unicode ドライバーについてはマッピングを行いません。

ANSI 関数から Unicode 関数へのマッピング

CLI Unicode 関数の構文は、それに対応する ANSI 関数の構文と同じですが、SQLCHAR パラメーターが SQLWCHAR として定義されている点は異なります。ANSI 構文で SQLPOINTER と定義されている文字バッファーは、 Unicode 関数では、SQLCHAR か SQLWCHAR のいずれかとして定義できます。 ANSI 構文の詳細は、ANSI バージョンの CLI Unicode 関数を参照してください。

SQLAllocConnect 関数 (CLI) - 接続ハンドルの割り振り

ODBC 3.0 では SQLAllocConnect() は使用すべきでない関数なので、代わりに SQLAllocHandle() を使用します。

CLI のこのバージョンでは引き続き SQLAllocConnect() がサポートされていますが、最新の標準に合わせて、SQLAllocHandle() を CLI プログラムでご使用になることをお勧めします。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

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

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

SQLAllocEnv 関数 (CLI) - 環境ハンドルの割り振り

ODBC 3.0 では SQLAllocEnv() は使用すべきでない関数なので、代わりに SQLAllocHandle() を使用します。

本バージョンの CLI では引き続き SQLAllocEnv() がサポートされていますが、最新の標準に合わせて、CLI プログラムでは SQLAllocHandle() を使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

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

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

SQLAllocHandle 関数 (CLI) - ハンドルの割り振り

環境、接続、ステートメント、または記述子ハンドルを割り振ります。

注: これは、ODBC 2.0 の使用すべきでない関数 SQLAllocConnect()、SQLAllocEnv()、および SQLAllocStmt() にとって代わる関数です。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

構文

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

関数引数

表 2. SQLAllocHandle 引数

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*HandleType*	入力	SQLAllocHandle() によって割り振られるハンドルのタイプ。以下の値のうちの 1 つでなければなりません。 • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
SQLHANDLE	*InputHandle*	入力	割り振られる新規ハンドルに対してコンテキストとして使用する既存のハンドル。 HandleType が SQL_HANDLE_ENV である場合、これは SQL_NULL_HANDLE になります。 HandleType が SQL_HANDLE_DBC の場合は、これは環境ハンドルでなければなりません。また HandleType が SQL_HANDLE_STMT または SQL_HANDLE_DESC である場合は、接続ハンドルでなければなりません。
SQLHANDLE *	*OutputHandlePtr*	出力	バッファーを指すポインター。そのバッファーの中で、新規に割り振られたデータ構造にハンドルが返されます。

使用法

SQLAllocHandle() は、環境、接続、ステートメント、および記述子ハンドルを割り振るために使用します。有効な *InputHandle* が存在すれば、アプリケーションは複数の環境、接続、ステートメント、または記述子ハンドルをいつでも割り振ることができます。

アプリケーションが、**OutputHandlePtr* を既存の環境、接続、ステートメント、または記述子ハンドルに設定して SQLAllocHandle() を呼び出すと、 CLI はハンドルを上書きし、そのハンドルのタイプに適した新規のリソースが割り振られます。元

のハンドルに関連した CLI リソースには、何の変更も加えられません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

SQLAllocHandle() は、SQL_INVALID_HANDLE を戻した場合、出力引数が NULL ポインターでない限り、 *OutputHandlePtr* を *HandleType* の値に応じて SQL_NULL_HENV、SQL_NULL_HDBC、SQL_NULL_HSTMT、または SQL_NULL_HDESC に設定します。これでアプリケーションは、*InputHandle* 引数内のハンドルに関連付けられた診断データ構造から追加情報を得ることができます。

診断

表 3. SQLAllocHandle SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08003	接続がクローズされています。	HandleType 引数は SQL_HANDLE_STMT または SQL_HANDLE_DESC でしたが、 *InputHandle* 引数で指定された接続ハンドルはオープンされた接続をもっていませんでした。 CLI がステートメント・ハンドルまたは記述子ハンドルを割り振るには、接続処理が正常に完了して (そして接続がオープンされて) いなければなりません。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY013	予期しない、メモリーのハンドル・エラーです。	*HandleType* 引数が SQL_HANDLE_DBC、 SQL_HANDLE_STMT、または SQL_HANDLE_DESC であり、基礎メモリー・オブジェクトにアクセスできない (メモリー不足が原因と考えられる) ため、関数呼び出しを処理できませんでした。
HY014	これ以上ハンドルがありません。	*HandleType* 引数に指定されたハンドル・タイプ別に割り振り可能なハンドル数の限度に達したか、または場合によっては、新規のハンドルを正しく初期化するのに十分なシステム・リソースがありません。
HY092	オプション・タイプが範囲外です。	*HandleType* 引数は、以下のいずれでもありませんでした。 • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC

制限

なし。

例

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

/* ... */

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

/* ... */

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

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

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

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

SQLAllocStmt 関数 (CLI) - ステートメント・ハンドルの割り振り

ODBC 3.0 では SQLAllocStmt() は使用すべきでない関数なので、代わりに SQLAllocHandle() を使用します。

このバージョンの CLI でも引き続き SQLAllocStmt() をサポートしていますが、最新の標準に準拠するように、SQLAllocHandle() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

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

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

SQLBindCol 関数 (CLI) - アプリケーション変数または LOB ロケーターへの列のバインド

アプリケーションは、結果セット内の列を C データ・タイプ変数に関連付け (バインド) したり、結果セット内の LOB 列を LOB ロケーターに関連付け (バインド) したりできます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

SQLBindCol() は、次のどちらかに結果セット内の列を関連付けるために使用します。

- すべての C データ・タイプのための、アプリケーション変数またはアプリケーション変数の配列 (ストレージ・バッファー)。 SQLFetch() または SQLFetchScroll() が呼び出されると、データがアプリケーションから DBMS へ転送されます。データが転送されると、データ変換が行われる可能性があります。
- LOB ロケーター (LOB 列用)。 SQLFetch() が呼び出されると、データそのものではなく LOB ロケーターが DBMS からアプリケーションへ転送されます。

  別の方法として、 SQLBindFileToCol() を使用して LOB 列をファイルに直接バインドすることができます。

SQLBindCol() は、アプリケーションが取り出す必要のある結果セット中の列ごとに 1 回呼び出されます。

一般的に、SQLPrepare()、SQLExecDirect()、またはスキーマ関数のうちの 1 つがこの関数の前に呼び出され、その後で SQLFetch()、SQLFetchScroll()、SQLBulkOperations()、または SQLSetPos() が呼び出されます。 SQLBindCol() を呼び出す前に列属性も必要となる場合があり、 SQLDescribeCol() または SQLColAttribute() を使用して取得することができます。

構文

```
SQLRETURN   SQLBindCol (
               SQLHSTMT          StatementHandle,          /* hstmt */
               SQLUSMALLINT      ColumnNumber,             /* icol */
               SQLSMALLINT       TargetType,               /* fCType */
               SQLPOINTER        TargetValuePtr,           /* rgbValue */
               SQLLEN            BufferLength,             /* dbValueMax */
               SQLLEN            *StrLen_or_IndPtr);       /* *pcbValue */
```

関数引数

表 4. SQLBindCol 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*ColumnNumber*	入力	列を識別する番号。列は、左から右へ順番に番号が付けられています。 • ブックマークを使用していない場合 (SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF)、列番号は 1 から始まります。 • ブックマークを使用している場合 (そのステートメント属性が SQL_UB_ON)、列番号は 0 から始まります。列 0 はブックマーク列です。

表 4. SQLBindCol 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*TargetType*	入力	結果セット内の列番号 *ColumnNumber* 用の C データ・タイプ。アプリケーションは、データ・ソースからデータを取り出してから、そのデータをこの C タイプに変換します。 `SQLBulkOperations()` または `SQLSetPos()` を使用すると、データ・ソースへの情報の送信時にドライバーがデータを C データ・タイプから変換します。以下のタイプがサポートされます。 • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DECIMAL_IBM • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_NUMERIC [a] • SQL_C_SBIGINT • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TYPE_TIMESTAMP_EXT • SQL_C_TINYINT • SQL_C_UBIGINT • SQL_C_UTINYINT • SQL_C_WCHAR SQL_C_DEFAULT を指定すると、データはデフォルトの C データ・タイプに転送されます。

表 4. SQLBindCol 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLPOINTER	*TargetValuePtr*	入出力 (据え置き)	フェッチが起きたときに、 CLI が列方向または行方向のバインディングを使って列データまたは LOB ロケーターを保管するバッファーまたはバッファー配列を指すポインター。 *Operation* 引数 SQL_REFRESH を使用する関数 `SQLFetch()`、`SQLFetchScroll()`、`SQLSetPos()` の呼び出しか、または *Operation* 引数 SQL_FETCH_BY_BOOKMARK を使用する関数 `SQLBulkOperations()` の呼び出し時にデータを戻すのにこのバッファーが使われます。それ以外では `SQLBulkOperations()` と `SQLSetPos()` は、データを検索するのにバッファーを使います。 *TargetValuePtr* が NULL の場合は、列はアンバインドされます。 SQL_UNBIND オプションを使って `SQLFreeStmt()` を呼び出せば、どの列でもアンバインドすることができます。
SQLLEN	*BufferLength*	入力	列データまたは LOB ロケーターを保管するのに使用できる *TargetValuePtr* バッファーのサイズ (バイト単位)。 *TargetType* が、バイナリーまたは文字ストリング (単一バイトまたは 2 バイト) を表す場合や、可変長データを戻す列の SQL_C_DEFAULT である場合、 *BufferLength* は 0 より大きくなければならず、そうでないとエラーが返されます。文字データの場合、ドライバーは NULL 終止符文字をカウントするので、それに合わせてスペースを割り振る必要があることに注意してください。他のすべてのデータ・タイプの場合、この引数は無視されます。

表 4. SQLBindCol 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLLEN *	*StrLen_or_IndPtr*	入出力 (据え置き)	CLI が *TargetValuePtr* バッファー内に返すために使用できるバイト数を示す値 (または値の配列) を指すポインター。 *TargetType* が LOB ロケーターの場合は、 LOB データのサイズではなくロケーターのサイズが返されます。 *Operation* 引数 SQL_REFRESH を使用する関数 SQLFetch()、SQLFetchScroll()、SQLSetPos() の呼び出しか、または *Operation* 引数 SQL_FETCH_BY_BOOKMARK を使用する関数 SQLBulkOperations() の呼び出し時にデータを戻すのにこのバッファーが使われます。それ以外では SQLBulkOperations() と SQLSetPos() は、データを検索するのにバッファーを使います。 列のデータ値が NULL の場合は、 SQLFetch() はこの引数に SQL_NULL_DATA を返します。 このポインターの値は、バインドされた列ごとにユニークであるか、または NULL でなければなりません。 SQLBulkOperations() で使えるように、値 SQL_COLUMN_IGNORE、 SQL_NTS、SQL_NULL_DATA、またはデータの長さを設定することができます。 SQL_NO_LENGTH が返されることもあります。詳細については、使用法の項を参照してください。

- この関数の場合、 *TargetValuePtr* および *StrLen_or_IndPtr* のどちらも据え置き出力です。それは、結果セット行がフェッチされるまでは、これらのポインターが指す保管ロケーションは更新されないことを意味します。したがって、これらのポインターで参照される保管ロケーションは、 SQLFetch() または SQLFetchScroll() が呼び出されるまで有効でなければなりません。例えば、SQLBindCol() をローカル関数内に呼び出す場合、その関数の同じ有効範囲内から SQLFetch() を呼び出すか、または *TargetValuePtr* バッファーを静的バッファーまたはグローバル・バッファーとして割り振る必要があります。
- メモリー内で *TargetValuePtr* が *StrLen* のすぐ後に連続していると、 CLI はすべての可変長データ・タイプのデータ検索を最適化することができます。

使用法

結果セット内の列用にデータまたは LOB ロケーター (LOB 列の場合) を検索するときに、その列ごとに 1 回ずつ SQLBindCol() を呼び出します。 SQLFetch() または SQLFetchScroll() を呼び出して結果セットからデータを取り出すと、バインドされた各列内のデータは、 *TargetValuePtr* および *StrLen_or_IndPtr* ポインターによって割り当てられたロケーション内に置かれます。ステートメント属性 SQL_ATTR_ROW_ARRAY_SIZE が 1 より大きい場合、 *TargetType* はバッファー

配列を参照していなければなりません。 *TargetType* が LOB ロケーターの場合は、実際の LOB データではなくロケーター値が返されます。 LOB ロケーターは LOB 列内のデータ値全体を参照します。

CLI アプリケーションが関数 SQLBindCol() を使用して LOB 列に出力バッファーを提供していない場合、IBM® Data Server Client はデフォルトで、結果セットの中の LOB 列ごとに、アプリケーションに代わって LOB ロケーターを要求します。

列は、左から右へ順番に割り当てられた番号で識別されます。
- ブックマークを使用していない場合 (SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF)、列番号は 1 から始まります。
- ブックマークを使用している場合 (そのステートメント属性が SQL_UB_ON)、列番号は 0 から始まります。

列のバインドが完了した後のフェッチでアプリケーションは SQLBindCol() を呼び出して、これらの列のバインドの変更や、すでにアンバインドされている列のバインドを行うことができます。新規のバインドは、すでにフェッチしたデータには適用されず、次回のフェッチから使用されます。単一の列 (SQLBindFileToCol() を使用してバインドされている列を含む) をアンバインドするには、 NULL に設定した *TargetValuePtr* ポインターを用いて SQLBindCol() を呼び出します。すべての列をアンバインドするには、アプリケーションは、 SQL_UNBIND に設定してある *Option* 入力を用いて SQLFreeStmt() を呼び出します。

アプリケーションは、取り出されるデータのために十分なストレージが割り振られていることを確認する必要があります。バッファーに可変長のデータを入れる場合、バインドされた列の最大長と同じ大きさのストレージに NULL 終止符文字を加えた容量をアプリケーションで割り当てる必要があります。そうしないと、データは切り捨てられることがあります。バッファーに固定長データを入れる場合、 CLI はバッファーのサイズを C データ・タイプの長さと想定します。データ変換を指定すると、必須サイズが変わる場合があります。

ストリングの切り捨てが起きた場合、SQL_SUCCESS_WITH_INFO が返されて、*StrLen_or_IndPtr* が、アプリケーションへの戻りに使用できる実際のサイズの *TargetValuePtr* に設定されます。

切り捨ても SQL_ATTR_MAX_LENGTH ステートメント属性によって影響を受けます (そのステートメント属性はアプリケーションへ返されるデータ量を制限するために使用されます)。アプリケーションは、 SQL_ATTR_MAX_LENGTH とすべての可変長列に返される最大長の値を使用して SQLSetStmtAttr() を呼び出し、さらに、同じサイズ (および NULL 終止符文字分) の *TargetValuePtr* バッファーを割り振ることによって、切り捨てを報告しないように指定することができます。列データが設定された最大長より大きい場合、値がフェッチされたときに SQL_SUCCESS が返されますが、実際の長さではなく最大長が *StrLen_or_IndPtr* に返されます。

バインドされる列が SQL_GRAPHIC、SQL_VARGRAPHIC、または SQL_LONGVARGRAPHIC のタイプである場合は、*TargetType* を SQL_C_DBCHAR または SQL_C_CHAR に設定することができます。 *TargetType* が SQL_C_DBCHAR である場合は、 *TargetValuePtr* バッファーにフェッチされるデータは、 2 バイトの NULL 終止符文字でヌル終了します。 *TargetType* が SQL_C_CHAR である場合は、データのヌル終了はありません。いずれの場合も、

TargetValuePtr バッファーの長さ (*BufferLength*) はバイト単位であり、そのため 2 の倍数になります。PATCH1 キーワードを使って CLI に GRAPHIC ストリングを強制的にヌル終了させることもできます。

注: SQL_NO_TOTAL は、次の場合に *StrLen_or_IndPtr* に返されます。
- SQL タイプが可変長タイプであり、かつ
- *StrLen_or_IndPtr* と *TargetValuePtr* が連続しており、かつ
- 列タイプが NOT NULLABLE (NULL 不可) であり、かつ
- ストリング切り捨てが起きているとき。

記述子および SQLBindCol

以下の項では、SQLBindCol() が記述子と対話する方法について説明します。

注: 1 つのステートメントに SQLBindCol() 呼び出しを行うと、他のステートメントにも影響します。それが生じるのは、ステートメントに関連した ARD が明示的に割り当てられ、それらが他のステートメントにも関連しているような場合です。SQLBindCol() は記述子を変更するので、そのような変更は、この記述子が関連しているすべてのステートメントに適用されます。これが必要な動作でない場合、アプリケーションは、 SQLBindCol() を呼び出す前に、他のステートメントからこの記述子の関連付けを解除する必要があります。

引数のマッピング

概念的には、SQLBindCol() は、以下のステップを順次実行します。
- SQLGetStmtAttr() を呼び出して、ARD ハンドルを獲得します。
- SQLGetDescField() を呼び出して、この記述子の SQL_DESC_COUNT フィールドを獲得し、そして *ColumnNumber* 引数の値が SQL_DESC_COUNT の値を超える場合は、 SQLSetDescField() を呼び出して、 SQL_DESC_COUNT の値を *ColumnNumber* に増やします。
- SQLSetDescField() を複数回呼び出して、値を ARD の以下のフィールドに割り当てます。
  - SQL_DESC_TYPE と SQL_DESC_CONCISE_TYPE を *TargetType* の値に設定します。
  - *TargetType* に応じて、1 つ以上の SQL_DESC_LENGTH、SQL_DESC_PRECISION、SQL_DESC_SCALE を設定します。
  - SQL_DESC_OCTET_LENGTH フィールドを *BufferLength* の値に設定します。
  - SQL_DESC_DATA_PTR フィールドを *TargetValue* の値に設定します。
  - SQL_DESC_INDICATOR_PTR フィールドを *StrLen_or_IndPtr* の値に設定します (以下の節を参照してください)。
  - SQL_DESC_OCTET_LENGTH_PTR フィールドを *StrLen_or_IndPtr* の値に設定します (以下の節を参照してください)。

StrLen_or_IndPtr 引数が参照する変数は、標識および長さの情報のために使用されます。フェッチがその列について NULL 値を検出した場合は、この変数に SQL_NULL_DATA を保管し、そうでなければ、この変数にデータ長を保管します。 NULL ポインターを *StrLen_or_IndPtr* として渡せば、フェッチ操作でデータ長を返さないようにできますが、 NULL 値を検出した場合に SQL_NULL_DATA を返す方法がないと、フェッチは失敗してしまいます。

SQLBindCol() の呼び出しが失敗したときは、それが設定するはずであった ARD 内の記述子の内容フィールドは未定義になり、 ARD の SQL_DESC_COUNT フィールドの値は未変更のままになります。

COUNT フィールドの暗黙的なリセット

SQLBindCol() は、SQL_DESC_COUNT を *ColumnNumber* 引数の値に設定します。これを行うのは、SQL_DESC_COUNT の値を増加させることになるときだけです。*TargetValuePtr* 引数の値が NULL ポインターで、 *ColumnNumber* 引数の値が SQL_DESC_COUNT と等しいならば (すなわち、最も高いバインド列をアンバインドするとき)、SQL_DESC_COUNT は、最も高い残りのバインド列の数値に設定されます。

SQL_C_DEFAULT についての注意

列データを正常に取り出すには、アプリケーションは、アプリケーション・バッファー内にあるデータの長さと開始点を正確に判別しなければなりません。アプリケーションが明示的な *TargetType* を指定するならば、アプリケーションの誤解は容易に検出されます。ただしアプリケーションが SQL_C_DEFAULT の *TargetType* を指定した場合、メタデータに変更を加えるか、または異なる列にコードを適用することで、アプリケーションで予定していた列とは異なるデータ・タイプの列に対して SQLBindCol() を適用することができます。この場合、アプリケーションは、フェッチする列データの開始または長さを判別するのに失敗する可能性があります。このことで、報告されないデータ・エラーまたはメモリー違反が起きることがあります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 5. SQLBindCol SQLSTATEs

SQLSTATE	説明	解説
07009	無効な記述子索引	引数 *ColumnNumber* に指定された値が、結果セット内の列の最大数を超えたか、あるいは指定した値が 0 より小さいです。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY003	プログラム・タイプが範囲外です。	*TargetType* は、有効なデータ・タイプまたは SQL_C_DEFAULT ではありませんでした。

表 5. SQLBindCol SQLSTATEs (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* に指定された値は 1 より小さく、引数 *TargetType* は SQL_C_CHAR、 SQL_C_BINARY、または SQL_C_DEFAULT のいずれかです。
HYC00	ドライバーが使用できません。	CLI は引数 *TargetType* に指定されているデータ・タイプを認識はしますが、サポートしません。 LOB ロケーター C データ・タイプが指定されましたが、接続されているサーバーは LOB データ・タイプをサポートしていません。

注: フェッチ時に、バインドされた列に関係する付加的な診断メッセージが報告されることがあります。

制限

LOB データ・サポートは、ラージ・オブジェクト・データ・タイプをサポートするサーバーに接続しているときしか使用できません。 LOB ロケーター C データ・タイプをサポートしないサーバーに対してアプリケーションがそのデータ・タイプを指定した場合、 SQLSTATE **HYC**00 が返されます。

例

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

SQLBindFileToCol 関数 (CLI) - LOB 列への LOB ファイル参照のバインド

結果セット内の LOB または XML 列を 1 つのファイル参照またはファイル参照の配列に関連付けたりバインドしたりします。

この処理により、ステートメント・ハンドル用に各行がフェッチされた時点で、その列のデータをファイルへ直接転送することができます。

仕様:

- **CLI** 2.1

LOB ファイル参照引数 (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内 (クライアント上) のファイルを参照します。各行をフェッチする前に、アプリケーション側でこれらの変数にファイル名、ファイル名の長さ、およびファイル・オプション (新規/上書き/追加) が含まれていることを確認する必要があります。これらの値は、各行のフェッチ操作のたびに変更することができます。

構文

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

関数引数

表 6. SQLBindFileToCol 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*icol*	入力	列を識別する番号。列は、1 を最初の番号として順次左から右へ番号が付けられます。
SQLCHAR *	*FileName*	入力 (据え置き)	次回のフェッチの時に *StatementHandle* を使用して、ファイル名またはファイル名の配列を入れる場所を指すポインター。これは、ファイルの完全パス名か相対ファイル名のどちらかです。相対ファイル名の場合は、その名前は実行中のアプリケーションの現行パスに付加されます。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)	次回のフェッチの時に *StatementHandle* を使用して、ファイル名の長さ (またはファイル名の長さの配列) を入れるロケーションを指すポインター。このポインターが NULL の場合、*FileName* は、 SQL_NTS の長さの引き渡しと同様にヌル終了ストリングと見なされます。 ファイル名の長さの最大値は 255 です。

表 6. SQLBindFileToCol 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLUINTEGER *	*FileOptions*	入力 (据え置き)	次回のフェッチの時に *StatementHandle* を用いて、ファイルの書き込みを行うときに使用するファイル・オプション (またはファイル・オプションの配列) を入れるロケーションを指すポインター。以下の *FileOptions* がサポートされます。 **SQL_FILE_CREATE** 新しいファイルを作成します。この名前のファイルがすでに存在する場合は、SQL_ERROR が返されます。 **SQL_FILE_OVERWRITE** ファイルがすでに存在する場合は、そのファイルを上書きします。存在しない場合は、新しいファイルを作成します。 **SQL_FILE_APPEND** ファイルがすでに存在する場合は、そのファイルにデータを付加します。存在しない場合は、新しいファイルを作成します。 1 つのファイルに対してオプションは 1 つしか選択できず、デフォルト値はありません。
SQLSMALLINT	*MaxFileNameLength*	入力	これは *FileName* バッファーの長さを指定するか、またはアプリケーションが SQLFetchScroll() を使用して LOB 列用に複数行を取り出す場合は *FileName* 配列内の各エレメントの長さを指定します。
SQLINTEGER *	*StringLength*	出力 (据え置き)	返される LOB データのバイト単位の長さ (または長さの配列) を入れるロケーションを指すポインター。ポインターが NULL の場合は、何も返されません。
SQLINTEGER *	*IndicatorValue*	出力 (据え置き)	標識値 (または標識値の配列) を入れるロケーションを指すポインター。

使用法

アプリケーションは、行がフェッチされるときにファイルへ直接転送しなければならない列ごとに 1 回ずつ SQLBindFileToCol() を呼び出します。 LOB データは、データ変換および NULL 終止符文字の追加をせずに、ファイルに直接書き込まれます。 XML データは UTF-8 で書き出されます。このとき、SQL_ATTR_XML_DECLARATION 接続の設定またはステートメント属性に従って生成された XML 宣言を伴います。

FileName、 *FileNameLength*、および *FileOptions* は各フェッチの前に設定しなければなりません。 SQLFetch() または SQLFetchScroll() が呼び出されるとき、 LOB ファイル参照にバインドされている列のデータは、ファイル参照によって指されているファイル (複数を含む) に書き込まれます。フェッチの際に、SQLBindFileToCol() の据え置き入力引数値に関連したエラーが報告されます。 LOB ファイル参照、および据え置かれた *StringLength* および *IndicatorValue* 出力引数は、フェッチ操作が行われるたびに更新されます。

SQLFetchScroll() を使用して LOB 列用の複数行を取り出す場合、*FileName*、*FileNameLength*、 および *FileOptions* は、 LOB ファイル参照変数の配列を指します。この場合、*MaxFileNameLength* は *FileName* 配列内の各エレメントの長さを指定し、 CLI が *FileName* 配列内の各エレメントのロケーションを判別するのに使用されます。ファイル参照の配列の内容は、SQLFetchScroll() 呼び出しの時に有効でなければなりません。 *StringLength* および *IndicatorValue* ポインターは、それぞれ SQLFetchScroll() 呼び出しの際に更新されているエレメントの配列を指します。

SQLFetchScroll() を使用して、指定されたファイル名に従い、複数のファイルまたは同一のファイルに複数の LOB 値を書き込むことができます。同一ファイルに書き込む場合、ファイル名を入力するたびに SQL_FILE_APPEND ファイル・オプションを指定する必要があります。ファイル参照の配列の列方向バインドは、SQLFetchScroll() を使用する場合だけサポートされます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 7. SQLBindFileToCol SQLSTATE

SQLSTATE	説明	解説
07009	無効な列数です。	引数 *icol* に指定された値は、1 より小さい値でした。 引数 *icol* に指定された値が、データ・ソースでサポートされる列の最大数を超えました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	*FileName*、*StringLength*、または *FileOptions* は NULL ポインターです。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *MaxFileNameLength* に指定された値は、 0 より小さい値でした。
HYC00	ドライバーが使用できません。	アプリケーションは現在、ラージ・オブジェクトをサポートしないデータ・ソースに接続しています。

制限

この関数は、ラージ・オブジェクト・データ・タイプをサポートしない DB2 サーバーに接続されている場合には使用できません。関数タイプを SQL_API_SQLBINDFILETOCOL に設定してある SQLGetFunctions() を呼び出し、*SupportedPtr* 出力引数を調べて、現行の接続でその関数がサポートされているかどうかを判別してください。

例

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

SQLBindFileToParam 関数 (CLI) - LOB パラメーターへの LOB ファイル参照のバインド

SQLBindFileToParam() を使用するのは、 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の配列に関連付けたりバインドしたりする場合です。これで、以後のこのステートメントの実行時に、そのファイルのデータを LOB または XML 列に直接転送できるようになります。

仕様:

- **CLI** 2.1

LOB ファイル参照引数 (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内 (クライアント上) のファイルを参照します。SQLExecute() または SQLExecDirect() を呼び出す前に、アプリケーションはこの情報が据え置き入力バッファーで使用できるかどうかを確認する必要があります。これらの値は、SQLExecute() 呼び出しの間に変更することができます。

構文

```
SQLRETURN SQLBindFileToParam (
             SQLHSTMT          StatementHandle,          /* hstmt */
             SQLUSMALLINT      TargetType,               /* ipar */
             SQLSMALLINT       DataType,                 /* fSqlType */
             SQLCHAR           *FileName,
             SQLSMALLINT       *FileNameLength,
             SQLUINTEGER       *FileOptions,
             SQLSMALLINT       MaxFileNameLength,
             SQLINTEGER        *IndicatorValue);
```

関数引数

表 8. SQLBindFileToParam 引数

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

表 8. SQLBindFileToParam 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLUSMALLINT	*TargetType*	入力	パラメーター・マーカー番号。パラメーターは、1 を最初の番号として順次左から右へ番号が付けられます。
SQLSMALLINT	*DataType*	入力	列の SQL データ・タイプ。データ・タイプは、次のうちの 1 つでなければなりません。 • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_XML
SQLCHAR *	*FileName*	入力 (据え置き)	ステートメント (*StatementHandle*) が実行されるときに、ファイル名またはファイル名の配列を入れる場所を指すポインター。これは、ファイルの完全パス名か相対ファイル名のどちらかです。相対ファイル名の場合は、その名前はクライアント・プロセスの現行パスに付加されます。 この引数を NULL にすることはできません。
SQLSMALLINT *	*FileNameLength*	入力 (据え置き)	次回の *StatementHandle* を介する `SQLExecute()` または `SQLExecDirect()` の際に、ファイル名の長さ (またはファイル名の長さの配列) を入れるロケーションを指すポインター。 このポインターが NULL の場合、*FileName* は、SQL_NTS の長さの引き渡しと同様にヌル終了ストリングと見なされます。 ファイル名の長さの最大値は 255 です。
SQLUINTEGER *	*FileOptions*	入力 (据え置き)	ファイルの読み取りを行うときに、ファイル・オプション (またはファイル・オプションの配列) を入れるロケーションを指すポインター。このロケーションは、ステートメント (*StatementHandle*) を実行するときにアクセスされます。オプションは 1 つしかサポートされません (また、そのオプションを指定する必要があります)。 **SQL_FILE_READ** オープン、読み取り、クローズを行える正規ファイル。 (ファイルをオープンすると、長さが計算されます。) このポインターを NULL にすることはできません。
SQLSMALLINT	*MaxFileNameLength*	入力	これは、*FileName* バッファーの長さを指定します。アプリケーションが `SQLParamOptions()` を呼び出して各パラメーターに複数値を指定する場合、これは *FileName* 配列内の各エレメントの長さになります。

表 8. SQLBindFileToParam 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER *	*IndicatorValue*	入力 (据え置き)	標識値 (または標識値の配列) が入るロケーションを指すポインターで、パラメーターのデータ値が NULL になる場合は、 SQL_NULL_DATA に設定されます。データ値が NULL でない場合は、データ値を 0 にしなければなりません (または、ポインターを NULL に設定できます)。

使用法

アプリケーションは、ステートメントが実行されるときにファイルから直接取得しなければならない値を持つパラメーター・マーカーごとに 1 回ずつ SQLBindFileToParam() を呼び出します。ステートメントを実行する前に、*FileName*、*FileNameLength*、および *FileOptions* 値を設定しなければなりません。ステートメントが実行されると、 SQLBindFileToParam() を使用してバインドされたパラメーターの値が参照ファイルから読み取られ、サーバーに渡されます。

アプリケーションが SQLParamOptions() を使用して各パラメーターに複数値を指定する場合、 *FileName*、*FileNameLength*、および *FileOptions* は、 LOB ファイル参照変数の配列を指します。この場合、*MaxFileNameLength* は *FileName* 配列内の各エレメントの長さを指定し、 CLI が *FileName* 配列内の各エレメントのロケーションを判別するのに使用されます。

LOB パラメーター・マーカーは、SQLBindFileToParam() を使用することによって入力ファイルに、または SQLBindParameter() を使用することによってストアード・バッファーに関連付ける (バインドする) ことができます。最新のバインド・パラメーター関数呼び出しは、有効なバインドのタイプを判別します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 9. SQLBindFileToParam SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY004	SQL データ・タイプが範囲外です。	*DataType* に指定されている値は、この関数呼び出しにとって有効な SQL タイプではありませんでした。

表 9. SQLBindFileToParam SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引数の値が無効です。	*FileName*、*FileOptions FileNameLength*、は、NULL ポインターです。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	入力引数 *MaxFileNameLength* に指定された値は、0 より小さい値でした。
HY093	無効なパラメーター数です。	引数 *TargetType* に指定された値は、1 より小さいか、またはサポートされるパラメーターの最大数より大きい値でした。
HYC00	ドライバーが使用できません。	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ていません。

制限

この関数は、ラージ・オブジェクト・データ・タイプをサポートしない DB2 サーバーに接続されている場合には使用できません。関数タイプを SQL_API_SQLBINDFILETOPARAM に設定して SQLGetFunctions() を呼び出し、*SupportedPtr* 出力引数を調べて、現行の接続でその関数がサポートされているかどうかを判別してください。

例

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

SQLBindParameter 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド

SQL ステートメント内のパラメーター・マーカーを、アプリケーション変数、アプリケーション変数の配列、または LOB ロケーターにバインドします。

仕様:

- **CLI** 2.1
- **ODBC** 2.0

SQLBindParameter() は、パラメーター・マーカーを以下のいずれかにバインドします。

- すべての C データ・タイプのための、アプリケーション変数またはアプリケーション変数の配列 (ストレージ・バッファー)。この場合、SQLExecute() または SQLExecDirect() が呼び出されると、データがアプリケーションから DBMS へ転送されます。データが転送されると、データ変換が行われる可能性があります。
- LOB ロケーター (SQL LOB データ・タイプの場合)。この場合、SQL ステートメントが実行されると、 LOB データ自体でなく LOB ロケーター値がアプリケーションからサーバーへ転送されます。

  別の方法として、 SQLBindFileToParam() を使用して LOB パラメーターをファイルに直接バインドすることもできます。

また、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を、そのパラメーターの入力または出力 (あるいはその両方) が行われる可能性のあるアプリケーションにバインドするのにもこの関数を使用しなければなりません。

構文

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

関数引数

表 10. SQLBindParameter 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*ParameterNumber*	入力	パラメーター・マーカーは、1 を最初の番号として順次左から右へ番号が付けられます。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*InputOutputType*	入力	パラメーターのタイプ。 IPD の SQL_DESC_PARAMETER_TYPE フィールドの値も、この引数に設定されます。サポートされるタイプは次のとおりです。 • 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 CALL でない SQL ステートメントに関連付けられるか、 CALL されるストアード・プロシージャーの入力パラメーターにマークを付けます。 ステートメントが実行されると、パラメーターのデータはサーバーに送られるので、 *StrLen_or_IndPtr* バッファーに SQL_NULL_DATA または SQL_DATA_AT_EXEC が入っていない限り、 *ParameterValuePtr* バッファーには 1 つ以上の有効な入力データ値が入っていなければなりません (値が、 SQLParamData() と SQLPutData() を介して送られる必要がある場合)。 • SQL_PARAM_INPUT_OUTPUT: パラメーター・マーカーは、呼び出された (CALL された) ストアード・プロシージャー内の入出力パラメーターに関連付けられます。 ステートメントが実行されると、パラメーターのデータはサーバーに送られるので、 *StrLen_or_IndPtr* バッファーに SQL_NULL_DATA または SQL_DATA_AT_EXEC が入っていない限り、 *ParameterValuePtr* バッファーには 1 つ以上の有効な入力データ値が入っていなければなりません (値が、 SQLParamData() と SQLPutData() を介して送られる必要がある場合)。 • SQL_PARAM_OUTPUT: パラメーター・マーカーは、呼び出された (CALL された) ストアード・プロシージャー内の出力パラメーターまたはストアード・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、出力パラメーターのデータは、 *ParameterValuePtr* および *StrLen_or_IndPtr* によって指定されるアプリケーション・バッファーに返されます。ただし、これは *ParameterValuePtr* および *StrLen_or_IndPtr* が NULL ポインターでない場合のことで、両者が NULL ポインターの場合は、出力データは廃棄されます。出力パラメーターが戻り値を持たない場合、 *StrLen_or_IndPtr* は SQL_NULL_DATA に設定されます。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	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ます。 • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DECIMAL_IBM • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_NUMERIC [a] • SQL_C_SBIGINT • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TYPE_TIMESTAMP_EXT • SQL_C_TYPE_TIMESTAMP_EXT_TZ • SQL_C_TINYINT • SQL_C_UBIGINT • SQL_C_UTINYINT • SQL_C_WCHAR SQL_C_DEFAULT を指定すると、データがデフォルトの C データ・タイプから *ParameterType* で指定するタイプに転送されることになります。 **a** Windows 32 ビットのみ

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*ParameterType*	入力	SQL データ・タイプのパラメーター。サポートされるタイプは次のとおりです。 • SQL_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_TYPE_TIMESTAMP_WITH_TIMEZONE • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR • SQL_XML **注:** SQL_BLOB_LOCATOR、SQL_CLOB_LOCATOR、 SQL_DBCLOB_LOCATOR は、アプリケーション関連の概念であり、 CREATE TABLE ステートメント中に列定義するためのデータ・タイプにはマッピングしません。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLULEN	*ColumnSize*	入力	対応するパラメーター・マーカーの精度。 *ParameterType* が以下を表している場合、次のようになります。 • バイナリーまたは 1 バイト文字ストリング (例えば、SQL_CHAR、SQL_BLOB) を示している場合、これはこのパラメーター・マーカーの最大長 (バイト数) です。 • 2 バイト文字ストリング (例えば、SQL_GRAPHIC) の場合、これはこのパラメーターに関する 2 バイト文字の最大長です。 • SQL_DECIMAL、SQL_NUMERIC の場合、これは、最大の 10 進数の精度です。 • 外部ルーチン引数用の XML 値 (SQL_XML) の場合、これは、宣言された XML AS CLOB(n) 引数の最大バイト数、n です。タイプ SQL_XML の他のすべてのデータ・タイプの場合、この引数は無視されます。 • それ以外の場合は、この引数は無視されます。
SQLSMALLINT	*DecimalDigits*	入力	*ParameterType* が SQL_DECIMAL または SQL_NUMERIC の場合、 *DecimalDigits* は、対応するパラメーターのスケールを表し、 IPD の SQL_DESC_SCALE フィールドを設定します。 *ParameterType* が SQL_TYPE_TIMESTAMP または SQL_TYPE_TIME の場合、 *Decimal Digits* は、対応するパラメーターの精度を表し、 IPD の SQL_DESC_PRECISION フィールドを設定します。時刻タイム・スタンプ値の精度は、時刻またはタイム・スタンプのストリング表示の小数点の右側の桁数です (例えば、 yyyy-mm-dd hh:mm:ss.fff のスケールは 3 になります)。 上記以外の *ParameterType* 値の場合、*DecimalDigits* は無視されます。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLPOINTER	*ParameterValuePtr*	入力 (据え置き)、 出力 (据え置き)、 あるいは その両方	• 入力時 (*InputOutputType* は SQL_PARAM_INPUT、または SQL_PARAM_INPUT_OUTPUT に設定されます) 実行時に、*StrLen_or_IndPtr* が SQL_NULL_DATA または SQL_DATA_AT_EXEC を含まない場合、*ParameterValuePtr* は、そのパラメーターの実際のデータがあるバッファーを指します。 *StrLen_or_IndPtr* が SQL_DATA_AT_EXEC を含む場合、 *ParameterValuePtr* は、このパラメーターに関連したアプリケーション定義の 32 ビット値です。この 32 ビット値は、以後の SQLParamData() 呼び出しによってアプリケーションに返されます。 SQLParamOptions() が呼び出されて、パラメーターの複数の値を指定する場合、*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。 • 出力時 (*InputOutputType* は SQL_PARAM_OUTPUT、または SQL_PARAM_INPUT_OUTPUT に設定されます) *ParameterValuePtr* は、ストアード・プロシージャーの出力パラメーター値が保管されるバッファーを指します。 *InputOutputType* を SQL_PARAM_OUTPUT に設定し、 *ParameterValuePtr* と *StrLen_or_IndPtr* がともに NULL ポインターである場合、ストアード・プロシージャー呼び出しからの出力パラメーター値または戻り値は廃棄されます。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLLEN	*BufferLength*	入力	文字データとバイナリー・データの場合、*BufferLength* は、 *ParameterValuePtr* バッファーの長さを指定するか (そのバッファーが単一エレメントとして扱われる場合)、または *ParameterValuePtr* 配列内の個々のエレメントの長さを指定します (アプリケーションが SQLParamOptions() を呼び出して各パラメーターに複数の値を指定する場合)。文字およびバイナリー以外のデータの場合、この引数は無視されます。つまり、*ParameterValuePtr* のバッファーの長さ (単一エレメントである場合) または *ParameterValuePtr* 配列内の各エレメントの長さ (SQLParamOptions() を使用して各パラメーターに値の配列を指定する場合) は、 C データ・タイプに関連した長さであることが前提とされます。 出力パラメーターの場合、*BufferLength* を使用して、以下の方法で文字またはバイナリー出力データを切り捨てるかどうかを判別します。 • 文字データの場合、戻りに使用できるバイト数が *BufferLength* 以上であれば、 *ParameterValuePtr* 内のデータは *BufferLength-1* バイトに切り捨てられ、ヌル終了します (ヌル終了がオフになっていない場合)。 • バイナリー・データの場合、戻りに使用できるバイト数が *BufferLength* より大きいと、*ParameterValuePtr* 内のデータが *BufferLength* バイトに切り捨てられます。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLLEN *	*StrLen_or_IndPtr*	入力 (据え置き)、 出力 (据え置き)、 あるいは その両方	これが入力または入出力パラメーターである場合: これは、 *ParameterValuePtr* に保管されているパラメーター・マーカー値の長さを (ステートメントの実行時に) 入れるロケーションを指すポインターです。 パラメーター・マーカーに NULL 値を指定するには、この保管ロケーションに SQL_NULL_DATA を入れる必要があります。 *ValueType* が SQL_C_CHAR である場合、この保管ロケーションには、 *ParameterValuePtr* に保管されているデータの正確な長さか、または *ParameterValuePtr* の内容がヌル終了の場合は SQL_NTS が入っていなければなりません。 *ValueType* が (明示的に、または SQL_C_DEFAULT を使用して暗黙的に) 文字データを示し、かつこのポインターが NULL に設定されている場合、アプリケーションが常に *ParameterValuePtr* にヌル終了のストリングを提供すると想定されています。また、このことは、このパラメーター・マーカーが NULL 値ではありえないことをも暗黙指定しています。 *ParameterType* が GRAPHIC データ・タイプを表し、かつ *ValueType* が SQL_C_CHAR である場合は、*StrLen_or_IndPtr* を指すポインターは、NULL ではあり得ず、 *StrLen_or_IndPtr* の内容が SQL_NTS を保持することもあり得ません。一般に GRAPHIC データ・タイプの場合、この長さは 2 バイト・データが占有するオクテットの数であり、したがってこの長さは常に 2 の倍数になります。実際に、長さが奇数である場合には、ステートメントを実行しようとするとエラーが発生します。 SQLExecute() または SQLExecDirect() が呼び出され、かつ *StrLen_or_IndPtr* が SQL_DATA_AT_EXEC の値を指していると、パラメーターの値は SQLPutData() によって送信されます。このパラメーターは、**実行時データ・**パラメーターと呼ばれます。 SQLBindParameter() メソッドまたは SQLExtendedBind() メソッドが SQL_ATTR_EXTENDED_INDICATORS 属性を設定した後に呼び出される場合、*StrLen_or_IndPtr* 引数を使用すると、これらのメソッドを介して、SQL_UNASSIGNED 定数と SQL_DEFAULT_PARAM 定数を渡すことができます。

表 10. SQLBindParameter 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER *	StrLen_or_IndPtr *(cont)*	入力 (据え置き)、出力 (据え置き)、あるいはその両方	SQL_ATTR_PARAMSET_SIZE 属性を指定した SQLSetStmtAttr() を使用して各パラメーターに複数の値を指定すると、*StrLen_or_IndPtr* は、SQLINTEGER 値の配列 (ここでは NULL 終止符文字を除いてどのエレメントも、対応する *ParameterValuePtr* エレメント内のバイト数となり得る) を指すか、または SQL_NULL_DATA を指します。 *StrLen_or_IndPtr* は、パラメーターのサイズを表します。出力パラメーターがある場合、*StrLen_or_IndPtr* は SQLINTEGER に対するメモリー・アドレス (ポインター) で、値は以下のいずれかになります。 • バッファーの長さ (NULL 終止符の分を差し引いたもの)。 • -1 (SQL_NULL_DATA)。値が NULL であり、実際の値を無視できることを意味します。 • -4 (SQL_NO_TOTAL)。LOB タイプのデータ専用で、返すことができるバイト数を判別できないことを示すために使用されます。

使用法

SQLBindParameter() は、使用すべきでない SQLSetParam() 関数の機能を拡張して以下を行う手段になります。

- ストアード・プロシージャーのパラメーターを適切に処理するために必要なパラメーターが入力、入出力、または出力のどれであるかを指定します。
- SQL_ATTR_PARAMSET_SIZE 属性を指定した SQLSetStmtAttr() を SQLBindParameter() と一緒に使用するときに、入力パラメーター値の配列を指定します。

WHERE または UPDATE 節内のターゲット列のデータ・タイプと長さか、またはストアード・プロシージャーのパラメーターが分かっている場合、 SQLPrepare() より先にこの関数を呼び出すことができます。分からない場合は、SQLDescribeParam() を使ってステートメントを準備してから、パラメーター・マーカーをバインドした後で、ターゲット列またはストアード・プロシージャー・パラメーターの属性を取得することができます。

パラメーター・マーカーは、番号 (*ParameterNumber*) で参照されますが、 1 から始まって、左から右へ連続番号が付けられます。

ValueType によって指定された C バッファー・データ・タイプは、 *ParameterType* によって指定される SQL データ・タイプと互換性がなければならず、そうでない場合はエラーが発生します。

この関数によってバインドされたパラメーターはすべて、以下のいずれかのイベントが行われる時点まで有効です。

- SQLFreeStmt() に SQL_RESET_PARAMS オプションを指定して呼び出したとき。
- *HandleType* を SQL_HANDLE_STMT に設定して SQLFreeHandle() を呼び出すか、または SQL_DROP オプションを指定して SQLFreeStmt() を呼び出したとき。
- 同じ *ParameterNumber* で SQLBindParameter() を再度呼び出したとき。
- 関連した APD 記述子ハンドルを使って SQLSetDescField() を呼び出し、 APD のヘッダー・フィールド内の SQL_DESC_COUNT をゼロ (0) に設定したとき。

パラメーターは、ファイルまたは保管ロケーションのいずれか一方にのみバインドすることができ、その両方にバインドすることはできません。最新のバインド・パラメーター関数呼び出しによって、有効なバインドが決まります。

パラメーター・タイプ

InputOutputType 引数は、パラメーターのタイプを指定します。プロシージャーを呼び出さない SQL ステートメント内のパラメーターは、すべて入力パラメーターです。ストアード・プロシージャー呼び出しのパラメーターは、入力、入出力、または出力パラメーターにすることができます。通常、DB2 ストアード・プロシージャー引数の規則では、すべてのプロシージャー引数は入出力であることが前提になっていますが、アプリケーション・プログラマーは、選択によっては、入力や出力の特性を SQLBindParameter() 上でさらに詳しく指定し、より厳密なコーディング・スタイルに準じることができます。

- アプリケーション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は、 *InputOutputType* を SQL_PARAM_INPUT に設定し、データ・ソースがパラメーターの値を返したら、CLI がその値を廃棄します。
- アプリケーションがパラメーターに SQL_PARAM_INPUT_OUTPUT または SQL_PARAM_OUTPUT としてマークを付け、データ・ソースが値を返さない場合は、 CLI は *StrLen_or_IndPtr* バッファーを SQL_NULL_DATA に設定します。
- アプリケーションがパラメーターに SQL_PARAM_OUTPUT としてマークを付けると、そのパラメーターのデータは CALL ステートメントが処理された後にアプリケーションに返されます。 *ParameterValuePtr* および *StrLen_or_IndPtr* 引数が両方とも NULL ポインターである場合、 CLI は出力値を廃棄します。データ・ソースが出力パラメーターの値を返さない場合、 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* および *StrLen_or_IndPtr* は据え置き引数です。 *InputOutputType* が SQL_PARAM_INPUT または SQL_PARAM_INPUT_OUTPUT に設定されている場合、保管ロケーションは有効でなければならず、ステートメント実行時に入力データ値が入っていなければなりません。このことは、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に保持するか、あるいは、これらの保管ロケーションを動的に割り振るか静的またはグローバルに宣言しなければならないことを意味します。

  同様に、*InputOutputType* を SQL_PARAM_OUTPUT または SQL_PARAM_INPUT_OUTPUT に設定した場合は、 CALL ステートメントの実

行の完了時点まで *ParameterValuePtr* と *StrLen_or_IndPtr* のバッファー・ロケーションは有効のままでなければなりません。

ParameterValuePtr および StrLen_or_IndPtr 引数

ParameterValuePtr と *StrLen_or_IndPtr* は据え置き引数であるので、これらが指す保管ロケーションは、ステートメントの実行時には有効になっていてしかも入力データ値を収めていなければなりません。これは、SQLExecDirect() または SQLExecute() 呼び出しを、 SQLBindParameter() 呼び出しと同じアプリケーション関数有効範囲内に保つことを意味するか、または、その保管ロケーションの動的割り振りあるいはその保管ロケーションの静的またはグローバルな宣言を意味します。

ParameterValuePtr と *StrLen_or_IndPtr* で参照される変数内のデータはステートメントを実行するまで検査されないため、データの内容またはフォーマットのエラーは SQLExecute() または SQLExecDirect() を呼び出すまで検出も報告もされません。

アプリケーションは、パラメーターの値を、*ParameterValuePtr* バッファーに入れて、または SQLPutData() への 1 つ以上の呼び出しによって、渡すことができます。呼び出しを用いた場合、パラメーターは実行時データ・パラメーターになります。アプリケーションは、*StrLen_or_IndPtr* によって指し示されているバッファー内に SQL_DATA_AT_EXEC 値を置くことで、実行時データ・パラメーターを CLI に通知します。また、 *ParameterValuePtr* 入力引数を 32 ビット値に設定して、次の SQLParamData() 呼び出しの際に返されるようにし、パラメーター位置を表すのに使用できるようにします。

SQLBindParameter() を使用してアプリケーション変数をストアード・プロシージャーの出力パラメーターにバインドするときに、メモリー内で *StrLen_or_IndPtr* バッファーの後に連続して *ParameterValuePtr* バッファーを置くと、 CLI のパフォーマンスを若干向上させることができます。例:

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

BufferLength 引数

文字データおよびバイナリー C データの場合、 *ParameterValuePtr* バッファーが単一エレメントであれば、 *BufferLength* 引数はそのバッファーの長さを指定しますが、アプリケーションが SQL_ATTR_PARAMSET_SIZE 属性を指定して SQLSetStmtAttr() を呼び出して各パラメーターに複数値を指定したときには、*BufferLength* は、NULL 終止符文字を含めた *ParameterValuePtr* 配列内の*個々の* エレメントの長さになります。アプリケーションが複数値を指定する場合、*BufferLength* を使用して *ParameterValuePtr* 配列内の値のロケーションを判別します。その他の C データ・タイプの場合はすべて、*BufferLength* 引数は無視されます。

ColumnSize 引数

ターゲットの列または出力パラメーターの実サイズが分からない場合、アプリケーションは列の長さを 0 と指定しても問題ありません。 (*ColumnSize* を 0 に設定する)。

列のデータ・タイプが固定長の場合、 CLI ドライバーは長さをデータ・タイプそのものから判別します。しかし、データ・タイプが文字、バイナリー・ストリング、またはラージ・オブジェクトである場合に *ColumnSize* を 0 に設定すると、別の意味が生じます。

入力パラメーター

ColumnSize が 0 であると、CLI は列またはストアード・プロシージャー・パラメーターのサイズとして提供された SQL タイプの最大長を使用します。 CLI はそのサイズを使用して必要な変換をすべて実行します。

出力パラメーター (ストアード・プロシージャーのみ)

ColumnSize が 0 であると、 CLI はパラメーターのサイズとして *BufferLength* を使用します。注意が必要な点として、ストアード・プロシージャーは *BufferLength* のバイト数を超えるデータを戻してはならないということです。もし戻すと、打ち切り誤差が生じます。

入出力パラメーターの場合 (ストアード・プロシージャーのみ)

ColumnSize が 0 であると、 CLI は入力および出力の両方をターゲット・パラメーターとして *BufferLength* に設定します。その意味するところは、入力データはストアード・プロシージャーに送られる前に必要であればその新しいサイズに変換される一方で、最大限 *BufferLength* のバイト数のデータが戻されるはずであるということです。

必要でないかぎり、*ColumnSize* を 0 に設定することはお勧めしません。実行時に経費のかさむデータ長のチェックが CLI によって行われるからです。

記述子

パラメーターがどのようにバインドされるかは、APD および IPD のフィールドによって決まります。そのような記述子フィールドを設定するには、`SQLBindParameter()` 内で引数を使用します。そのフィールドは、`SQLSetDescField()` 関数で設定することもできます。ただし、`SQLBindParameter()` を呼び出すのにアプリケーションは記述子ハンドルを獲得する必要がないので、`SQLBindParameter()` を使ったほうが効率が高いです。

注: ある 1 つのステートメントで SQLBindParameter() を呼び出すと、他のステートメントにも影響を与える可能性があります。そのような事態が起きるのは、ステートメントに関連した APD が明示的に割り当てられていて、しかも他のステートメントにも関連している場合です。 APD のフィールドは SQLBindParameter() によって修正されるので、その修正は、この記述子が関連付けられているすべてのステートメントに適用されます。これが必須の動作でない場合、アプリケーションは、SQLBindParameter() を呼び出す前に、その他のステートメントとの記述子の関連付けを解除する必要があります。

概念的には、SQLBindParameter() は、以下のステップを順次実行します。

- SQLGetStmtAttr() を呼び出して、APD ハンドルを獲得します。
- SQLGetDescField() を呼び出して、 APD から SQL_DESC_COUNT ヘッダー・フィールドを獲得します。 *ParameterNumber* 引数の値が SQL_DESC_COUNT の値を超える場合は、 SQLSetDescField() を呼び出して、 SQL_DESC_COUNT の値を *ParameterNumber* に増やします。

- SQLSetDescField() を複数回呼び出して、値を APD の以下のフィールドに割り当てます。
  - SQL_DESC_TYPE および SQL_DESC_CONCISE_TYPE を *ValueType* の値に設定します。ただし、*ValueType* が日時のコンサイス ID の 1 つである場合は例外です。その場合は、SQL_DESC_TYPE を SQL_DATETIME に設定し、SQL_DESC_CONCISE_TYPE をコンサイス ID に設定し、そして SQL_DESC_DATETIME_INTERVAL_CODE を対応日時サブコードに設定します。
  - SQL_DESC_DATA_PTR フィールドを *ParameterValue* の値に設定します。
  - SQL_DESC_OCTET_LENGTH_PTR フィールドを *StrLen_or_Ind* の値に設定します。
  - SQL_DESC_INDICATOR_PTR フィールドも *StrLen_or_Ind* の値に設定します。

  StrLen_or_Ind パラメーターは、パラメーター値の標識情報および長さの両方を指定します。
- SQLGetStmtAttr() を呼び出して、IPD ハンドルを獲得します。
- SQLGetDescField() を呼び出して、IPD の SQL_DESC_COUNT フィールドを獲得します。 *ParameterNumber* 引数の値が SQL_DESC_COUNT の値を超える場合は、 SQLSetDescField() を呼び出して、SQL_DESC_COUNT の値を *ParameterNumber* に増やします。
- SQLSetDescField() を複数回呼び出して、IPD の以下のフィールドに値を割り当てます。
  - SQL_DESC_TYPE および SQL_DESC_CONCISE_TYPE を *ParameterType* の値に設定します。ただし、*ParameterType* が日時のコンサイス ID の 1 つである場合は例外です。その場合は、SQL_DESC_TYPE を SQL_DATETIME に設定し、 SQL_DESC_CONCISE_TYPE をコンサイス ID に設定し、そして SQL_DESC_DATETIME_INTERVAL_CODE を対応日時サブコードに設定します。
  - *ParameterType* に応じて、1 つ以上の SQL_DESC_LENGTH、SQL_DESC_PRECISION、および SQL_DESC_SCALE を設定します。

SQLBindParameter() への呼び出しが失敗したときは、APD に設定するはずの記述子の内容フィールドは未定義で、 APD の SQL_DESC_COUNT フィールドは変更されません。さらに、IPD 内の適当なレコードの SQL_DESC_LENGTH、SQL_DESC_PRECISION、SQL_DESC_SCALE、および SQL_DESC_TYPE フィールドは未定義で、 IPD の SQL_DESC_COUNT フィールドは変更されません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 11. SQLBindParameter SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	*ValueType* 引数によって識別されるデータ値から *ParameterType* 引数によって識別されるデータ・タイプへの変換は、意味のある変換ではありません。 (例えば、SQL_C_TYPE_DATE から SQL_DOUBLE への変換です。)
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY003	プログラム・タイプが範囲外です。	引数 *ParameterNumber* で指定された値が、有効なデータ・タイプまたは SQL_C_DEFAULT ではありません。
HY004	SQL データ・タイプが範囲外です。	引数 *ParameterType* に指定された値が、有効な SQL データ・タイプではありません。
HY009	引数の値が無効です。	引数 *ParameterValuePtr* は NULL ポインターで、引数 *StrLen_or_IndPtr* も NULL ポインターですが、 *InputOutputType* が SQL_PARAM_OUTPUT ではありません。
HY010	関数のシーケンス・エラーです。	SQLExecute() または SQLExecDirect() が SQL_NEED_DATA を返した後に関数が呼び出されましたが、すべての*実行時*データ・パラメーターのデータが送られたわけではありません。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY021	不整合な記述子情報	整合性チェック時にチェックされた記述子情報は、整合性がとれていませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* に指定された値は、0 より小さい値でした。
HY093	無効なパラメーター数です。	引数 *ValueType* に指定された値が、1 より小さいか、サーバーでサポートされる最大数より大きい値でした。
HY094	位取りの値が無効です。	*ParameterType* に指定された値が SQL_DECIMAL または SQL_NUMERIC であり、 *DecimalDigits* に指定された値が 0 より小さいかまたは引数 *ParamDef* (精度) の値より大きい値でした。 *ParameterType* に指定された値が SQL_C_TYPE_TIMESTAMP で、*ParameterType* に指定された値が SQL_CHAR または SQL_VARCHAR のどちらかであり、 *DecimalDigits* に指定された値が 0 より小さいかまたは 9 より大きい値でした。 *ParameterType* に指定された値が SQL_C_TIMESTAMP_EXT で、*ParameterType* に指定された値が SQL_CHAR または SQL_VARCHAR のどちらかであり、 *DecimalDigits* に指定された値が 0 より小さいかまたは 12 より大きい値でした。

表 11. SQLBindParameter SQLSTATE (続き)

SQLSTATE	説明	解説
HY104	精度の値が無効です。	*ParameterType* に指定された値が SQL_DECIMAL または SQL_NUMERIC のどちらかで、 *ParamDef* に指定された値が 1 より小さい値でした。
HY105	パラメーター・タイプが無効です。	*InputOutputType* が 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	ドライバーが使用できません。	CLI またはデータ・ソースが、引数 *ValueType* に指定された値と引数 *ParameterType* に指定された値との組み合わせによって指定された変換をサポートしません。 引数 *ParameterType* に指定された値が、 CLI またはデータ・ソースのどちらかでサポートされていません。

制限

CLI V5 およびそれ以降と、ODBC 2.0 およびそれ以降では、使用すべきでなくなった SQLSetParam() API は、SQLBindParameter() に置き換わっています。

StrLen_or_IndPtr 用のさらに別の値 SQL_DEFAULT_PARAM が ODBC 2.0 に導入されたので、アプリケーションから送信された値ではなく、パラメーターのデフォルト値をプロシージャーで使用することを指定できるようになりました。 DB2 ストアード・プロシージャー引数ではデフォルト値はサポートされないので、*StrLen_or_IndPtr* 引数にこの値を指定すると、 SQL_DEFAULT_PARAM 値は無効な長さとみなされ、 CALL ステートメントを実行したときにエラーになります。

また、ODBC 2.0 には、*StrLen_or_IndPtr* 引数を指定して使用する SQL_LEN_DATA_AT_EXEC (*length*) マクロも導入されました。このマクロは、後続の SQLPutData() 呼び出しを経由して文字またはバイナリー C データ用に送信されるデータ全体の長さの合計を指定するために使用されます。 DB2 ODBC ドライバーではこの情報の必要がないため、このマクロは必要ありません。 ODBC アプリケーションは、 SQL_NEED_LONG_DATA_LEN オプションを指定した SQLGetInfo() を呼び出して、ドライバーがこの情報を必要とするかどうかを調べます。 DB2 ODBC ドライバーは、 SQLPutData() にはこの情報は必要ないことを示す 'N' を戻します。

例

```
SQLSMALLINT parameter1 = 0;

/* ... */

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

SQLBrowseConnect 関数 (CLI) - データ・ソースへの接続に必要な属性の取得

データ・ソースへの接続に必要な属性および属性値を、反復して発見および列挙する方法をサポートします。

仕様:

- **CLI** 5.0
- **ODBC** 1

SQLBrowseConnect() への呼び出しごとにそれぞれ、属性および属性値の継承レベルを返します。すべてのレベルを列挙し終わると、データ・ソースへの接続が完了し、完全な接続ストリングが SQLBrowseConnect() によって返されます。SQL_SUCCESS または SQL_SUCCESS_WITH_INFO の戻りコードは、すべての接続情報が指定され、アプリケーションが今やデータ・ソースに接続されていることを示しています。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLBrowseConnectW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 12. SQLBrowseConnect 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLCHAR *	*InConnectionString*	入力	要求接続ストリングをブラウズします (InConnectionString 引数を参照)。
SQLSMALLINT	*InConnectionStringLength*	入力	**InConnectionString* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLCHAR *	*OutConnectionString*	出力	バッファーを指すポインター。そのバッファーの中にブラウズ結果の接続ストリングを返します (OutConnectionString 引数を参照)。
SQLSMALLINT	*OutConnectionString Capacity*	入力	**OutConnectionString* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。

表 12. SQLBrowseConnect 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*OutConnectionString LengthPtr*	出力	**OutConnectionString* 内に返すのに使用できる合計エレメント数 (NULL 終止符文字を除く)。戻り値に使用できるエレメント数が *OutConnectionStringCapacity* 以上の場合、 **OutConnectionString* 内の接続ストリングは、 *OutConnectionStringCapacity* から NULL 終止符文字分を差し引いた長さに切り捨てられます。

使用法

SQLBrowseConnect() は、割り当てられた接続を必要とします。SQLBrowseConnect() から SQL_ERROR が返されると、未解決の接続情報は廃棄され、その接続は未接続の状態に戻されます。

SQLBrowseConnect() が接続で初めて呼び出されるときは、ブラウズ要求の接続ストリングには DSN キーワードが入っていなければなりません。

SQLBrowseConnect() への各呼び出しの際に、アプリケーションはブラウズ要求の接続ストリングに接続属性値を指定します。 CLI は、連続したレベルの属性と属性値をブラウズ結果の接続ストリング内に返しますが、ブラウズ要求の接続ストリングにまだ列挙されていない接続属性がある限り、SQL_NEED_DATA を返します。アプリケーションは、ブラウズ結果の接続ストリングの内容を使用して、SQLBrowseConnect() への次の呼び出し用のブラウズ要求接続ストリングを作成します。すべての必須属性 (*OutConnectionString* 引数内のアスタリスクが先頭に付いていない属性) は、 SQLBrowseConnect() への次回の呼び出しに含める必要があります。アプリケーションは、現在のブラウズ要求接続ストリングを構築する際に、以前のブラウズ結果の接続ストリングの内容全体を単にそのままコピーできないことに注意してください。すなわち、アプリケーションは、前のレベルで設定した属性に対して別の値を指定することはできません。

接続の全レベルとそれに関連した属性が列挙されたら、CLI が SQL_SUCCESS を返して、データ・ソースへの接続が完了し、完全な接続ストリングがアプリケーションに返されます。その接続ストリングは、SQLDriverConnect() 用の引数として SQL_DRIVER_NOPROMPT オプションと一緒に使用して、別の接続を確立するのに適しています。その完全な接続ストリングは、 SQLBrowseConnect() をあらためて呼び出すときに使用することはできません。もし SQLBrowseConnect() を再び呼び出すことになったとしたら、呼び出しのシーケンス全体を反復しなければならなくなります。

ブラウズの処理中にリカバリー可能の非致命的エラーが生じた場合、SQLBrowseConnect() は SQL_NEED_DATA も戻します。そのようなエラーが生じるのは、例えばアプリケーションが無効パスワードを指定した場合や、アプリケーションが無効な属性キーワードを指定した場合などです。 SQL_NEED_DATA が返された場合にブラウズ結果の接続ストリングが未変更であると、エラーが発生したということなので、アプリケーションは、ブラウズ時のエラーの SQLSTATE を返す SQLGetDiagRec() を呼び出すことができます。これでアプリケーションは、属性を修正してブラウズを続行できます。

アプリケーションは、いつでも SQLDisconnect() を呼び出してブラウズ処理を終了することができます。 CLI は、未解決の接続をすべて終了し、接続を非接続状態へ戻します。

InConnectionString 引数

ブラウズ要求の接続ストリングは、次の構文になります。

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

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

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

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

説明
- character-string には 0 個以上の SQLCHAR または SQLWCHAR エレメントが入れられます。
- identifier には 1 個以上の SQLCHAR または SQLWCHAR エレメントが入れられます。
- attribute-keyword は大文字小文字を区別しません。
- attribute-value は大文字小文字を区別することがあります。
- **DSN** キーワードの値はブランクのみでは成立しません。
- **NEWPWD** は、パスワード変更要求の一部として使用されます。アプリケーションは、`NEWPWD=anewpass;` などとして使用する新しいストリングを指定するか、または `NEWPWD=;` を指定して CLI ドライバーによって生成されるダイアログ・ボックスが新しいパスワードの入力を要求するようにすることができます。

接続ストリングと初期設定ファイルの文法上の理由から、**[]{}(),;?*=!@** 文字の入っているキーワードおよび属性値は避ける必要があります。システム情報の文法上、キーワードとデータ・ソース名には、円記号 (¥) を入れることができません。 CLI バージョン 2 の場合、**DRIVER** キーワードの前後に中括弧が必要です。

あるキーワードがブラウズ要求の接続ストリングの中で繰り返される場合、 CLI は、最初に現れたものの値を使用します。 **DSN** および **DRIVER** キーワードが同じブラウズ要求の接続ストリング内にある場合、 CLI は、最初に現れたキーワードの方を使用します。

OutConnectionString 引数

ブラウズ結果の接続ストリングは、接続属性のリストになっています。接続属性は、属性キーワードとそれに対応する属性値から成っています。ブラウズ結果の接続ストリングは、以下の構文になります。

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

attribute ::= [*]attribute-keyword=attribute-value

attribute-keyword ::= ODBC-attribute-keyword
| driver-defined-attribute-keyword

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

attribute-value ::= {attribute-value-list} | ?
(中括弧はリテラルであり、CLI によって返されます。)
attribute-value-list ::= character-string [:localized-character
string] | character-string [:localized-character string], attribute-value-list

説明

- character-string と localized-character string には、 0 個以上の SQLCHAR または SQLWCHAR エレメントが入れられます。
- identifier および localized-identifier のエレメント数は 1 つ以上です。 attribute-keyword は大文字小文字を区別しません。
- attribute-value は大文字小文字を区別することがあります。

接続ストリングと初期化ファイルの文法上の理由から、**[]{}(),;?*=!@** 文字の入っているキーワード、ローカライズ ID、および属性値は避ける必要があります。システム情報の文法上、キーワードとデータ・ソース名には、円記号 (¥) を入れることができません。

ブラウズ結果の接続ストリングの構文は、以下のセマンティック規則に従って使用されます。

- アスタリスク (*) が属性キーワードの前にある場合、属性はオプションであり、SQLBrowseConnect() への次回呼び出しの際は省略することができます。
- 属性キーワード **UID** および **PWD** には、 SQLDriverConnect() に定義されているのと同じ意味があります。
- DB2 データベースに接続するときは、**DSN**、**UID**、および **PWD** だけが必要になります。その他のキーワードは、指定することができますが、接続には影響ありません。
- ODBC 属性キーワードおよびドライバー定義の属性キーワードには、日本語化されたキーワードまたは使いやすくされたキーワードが含まれています。これは、ダイアログ・ボックス内のラベルとしてアプリケーションで使用できます。しかし、ブラウズ要求ストリングを CLI に渡すときは、 **UID**、**PWD**、または ID を単独で使用する必要があります。
- {attribute-value-list} には、対応する属性キーワードに対して有効な実際の値が列挙されます。中括弧 ({}) は、選択項目のリストを示していないことに注意してください。中括弧は CLI によって返されます。例えば、サーバー名のリストやデータベース名のリストであることもあります。
- attribute-value が単一の疑問符 (?) である場合、単一の値が属性キーワードに対応します。例えば、UID=JohnS; PWD=Sesame
- SQLBrowseConnect() のどの呼び出しでも、次のレベルの接続処理を充足するのに必要な情報だけが返されます。 CLI は、各呼び出しで常にコンテキストを判別できるように、状態情報を接続ハンドルに関連付けます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 13. SQLBrowseConnect SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	バッファー **OutConnectionString* は、ブラウズ結果の接続ストリング全部を返せるほど大きくなかったので、ストリングが切り捨てられました。バッファー **OutConnectionStringLengthPtr* には、切り捨てられなかったブラウズ結果の接続ストリングの長さが入っています。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S00	接続ストリング属性が無効です。	無効な属性キーワードが、ブラウズ要求の接続ストリング (*InConnectionString*) の中に指定されました。 (関数は SQL_NEED_DATA を返します。) 属性キーワードがブラウズ要求の接続ストリング (*InConnectionString*) の中で指定されましたが、現在の接続レベルにあてはまりません。 (関数は SQL_NEED_DATA を返します。)
01S02	オプション値が変更されました。	CLI は、`SQLSetConnectAttr()` 内の *ValuePtr* 引数に指定した値をサポートしておらず、類似した値を代用しました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08001	データ・ソースに接続できませんでした。	CLI がデータ・ソースとの接続を確立できませんでした。
08002	接続が使用中です。	指定された接続は、データ・ソースとの接続を確立するためにすでに使用されており、接続がまだオープンしています。
08004	アプリケーション・サーバーが、接続の確立を拒否しました。	データ・ソースが、インプリメンテーションで定義された理由により接続の確立を拒否しました。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、 CLI と接続を試行していたデータ・ソースとの間の通信リンクが失敗しました。
28000	許可指定が無効。	ブラウズ要求の接続ストリング (*InConnectionString*) に指定されているように、ユーザー ID または許可ストリングもしくはその両方が、データ・ソースにより定義されている制約に違反していました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 13. SQLBrowseConnect SQLSTATE (続き)

SQLSTATE	説明	解説
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *InConnectionStringLength* に指定された値は、 0 より小さい値で、SQL_NTS に等しくありませんでした。 引数 *OutConnectionStringCapacity* に指定された値は、0 より小さい値でした。

制限

なし。

例

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

/* ... */

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

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

  /* if inadequate connection information was provided, exit
     the program */
  if (cliRC == SQL_NEED_DATA)
  {
    printf("  You can provide other connection information "
           "here by setting connInStr¥n");
    break;
  }

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

SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、またはフェッチ

キーセット・ドリブン・カーソル上の行のセットを、追加、更新、削除、またはフェッチします。

仕様:

- **CLI** 6.0
- **ODBC** 3.0

キーセット・ドリブン・カーソル上で以下の操作を実行するには、SQLBulkOperations() を使用します。
- 新しい行を追加する
- 各行がブックマークによって識別される行のセットを更新する
- 各行がブックマークによって識別される行のセットを削除する
- 各行がブックマークによって識別される行のセットをフェッチする

構文

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

関数引数

表 14. SQLBulkOperations の引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLSMALLINT	*Operation*	入力	以下を実行する操作です。 • SQL_ADD • SQL_UPDATE_BY_BOOKMARK • SQL_DELETE_BY_BOOKMARK • SQL_FETCH_BY_BOOKMARK

使用法

アプリケーションは SQLBulkOperations() を使用して、キーセット・ドリブン・カーソル内の現行の照会に対応する基本表またはビューに対して以下の操作を実行します。
- 新しい行を追加する
- 各行がブックマークによって識別される行のセットを更新する
- 各行がブックマークによって識別される行のセットを削除する
- 各行がブックマークによって識別される行のセットをフェッチする

汎用アプリケーションは、必要なバルク操作がサポートされているかどうかを最初に確認する必要があります。それを行うために、SQLGetInfo() に SQL_DYNAMIC_CURSOR_ATTRIBUTES1 および SQL_DYNAMIC_CURSOR_ATTRIBUTES2 の *InfoType* を指定して呼び出すことができます (例えば、SQL_CA1_BULK_UPDATE_BY_BOOKMARK が戻されるかどうかを確認するため)。

SQLBulkOperations() を呼び出した後は、ブロック・カーソルの位置が定義されていません。アプリケーションは SQLFetchScroll() を呼び出してカーソル位置を設定しなければなりません。アプリケーションは、 *FetchOrientation* 引数として SQL_FETCH_FIRST、SQL_FETCH_LAST、 SQL_FETCH_ABSOLUTE、または SQL_FETCH_BOOKMARK を指定した SQLFetchScroll() だけを呼び出すようにしてください。アプリケーションが SQLFetch()、または *FetchOrientation* 引数として

SQL_FETCH_PRIOR、SQL_FETCH_NEXT、または SQL_FETCH_RELATIVE を指定した SQLFetchScroll() を呼び出した場合、カーソル位置は定義されません。

バルク操作 (SQLBulkOperations() への呼び出し) では、列を無視できます。これを行うには、SQLBindCol() を呼び出して列長/標識バッファー (*StrLen_or_IndPtr*) を SQL_COLUMN_IGNORE に設定します。これは SQL_DELETE_BY_BOOKMARK バルク操作には適用されません。

この関数を使用してバルク操作を行うときに行を無視できないので、アプリケーションが SQL_ATTR_ROW_OPERATION_PTR ステートメント属性を SQLBulkOperations() の呼び出し時に設定する必要はありません。

SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性が示すバッファーには、SQLBulkOperations() への呼び出しによって影響される行数が含まれています。

Operation 引数が SQL_ADD または SQL_UPDATE_BY_BOOKMARK であって、カーソルに関連した照会指定の選択リストに同じ列に対する複数の参照が含まれるとき、エラーが生成されます。

戻りコード

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

診断

表 15. SQLBulkOperations SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	*Operation* 引数は SQL_FETCH_BY_BOOKMARK だったので、 1 つ以上の列用に戻されたデータ・タイプ SQL_C_CHAR または SQL_C_BINARY のストリングまたはバイナリー・データでは、非ブランク文字または非 NULL のバイナリー・データが切り捨てられました。
01S07	無効な変換です。	*Operation* 引数は SQL_FETCH_BY_BOOKMARK で、アプリケーション・バッファーのデータ・タイプは SQL_C_CHAR や SQL_C_BINARY ではなく、1 つ以上の列用にアプリケーション・バッファーに戻されたデータは切り捨てられました。 (数値 C データ・タイプの場合、数の小数部分は切り捨てられました。時刻およびタイム・スタンプのデータ・タイプの場合、時刻の小数部分は切り捨てられました。) (関数は、SQL_SUCCESS_WITH_INFO を返します。)

表 15. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
07006	制限付きデータ・タイプ属性違反。	*Operation* 引数は SQL_FETCH_BY_BOOKMARK でしたが、SQLBindCol() の呼び出しで、結果セットにある列のデータ値を *TargetType* 引数に指定されたデータ・タイプに変換できませんでした。 *Operation* 引数は SQL_UPDATE_BY_BOOKMARK または SQL_ADD であり、アプリケーション・バッファー内のデータ値を結果セットにある列のデータ・タイプに変換できませんでした。
07009	無効な記述子索引	引数 *Operation* は、SQL_ADD でしたが、結果セット内の列数より大きい列番号で列がバインドされたか、または列番号が 0 より小さい値でした。
21S02	派生した表の程度が列リストと一致しません。	引数 *Operation* は SQL_UPDATE_BY_BOOKMARK で、どの列も更新不能でした。理由は、どの列もバインドされていないか、読み取り専用であるか、バインド済みの長さ/標識バッファーが SQL_COLUMN_IGNORE であったためです。
22001	ストリング・データの右側が切り捨てられました。	結果セット内の列に文字またはバイナリー値を割り当てたため、非ブランク文字 (文字の場合) または非 NULL 文字 (バイナリーの場合) またはバイトが切り捨てられました。
22003	範囲外の数値。	*Operation* 引数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK でしたが、結果セットの列への数値割り当てによって、数の整数部分 (小数部分ではなく) が切り捨てられました。 引数 *Operation* は SQL_FETCH_BY_BOOKMARK で、1 つ以上のバインド済み列に数値を戻したことで、有効数字を失った可能性があります。
22007	無効な日時フォーマット。	*Operation* 引数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK であり、結果セットの列に日付またはタイム・スタンプ値を代入した結果、年、月、または日フィールドが範囲外になりました。 引数 *Operation* は SQL_FETCH_BY_BOOKMARK であり、1 つ以上のバインド済み列用の日付またはタイム・スタンプ値の戻りで、年、月、または日フィールドが範囲外になった可能性があります。
22008	日付/時刻フィールドのオーバーフロー。	*Operation* 引数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK であり、結果セットの列に送られるデータに対して日時演算を実行した結果、日時フィールド (年、月、日、時、分、または秒フィールド) がフィールドの値の許容範囲を超えたか、または、グレゴリオ暦に基づく日時の法則に対して無効な値になりました。 *Operation* 引数は SQL_FETCH_BY_BOOKMARK であり、結果セットから検索されるデータに対して日時演算を実行した結果、日時フィールド (年、月、日、時、分、または秒フィールド) がフィールドの値の許容範囲を超えたか、または、グレゴリオ暦に基づく日時の法則に対して無効な値になりました。

表 15. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
22018	キャスト指定の文字値が無効。	*Operation* 引数は SQL_FETCH_BY_BOOKMARK、 C タイプは正確な数値か近似値または日時データ・タイプ、そして列の SQL タイプは文字データ・タイプでしたが、列内の値はバインドされた C タイプの有効なリテラルではありませんでした。 引数 *Operation* は SQL_ADD または SQL_UPDATE_BY_BOOKMARK、SQL タイプは正確な値または近似値、または日時データ・タイプ、C タイプは SQL_C_CHAR でしたが、列内の値はバインドされた SQL タイプの有効なリテラルではありませんでした。
23000	整合性制約違反。	*Operation* 引数は SQL_ADD、 SQL_DELETE_BY_BOOKMARK、または SQL_UPDATE_BY_BOOKMARK で、整合性制約の違反が生じました。 *Operation* 引数は SQL_ADD で、バインドされていない列は NOT NULL と定義されていて、デフォルト値がありません。 *Operation* 引数は SQL_ADD で、バインドされた *StrLen_or_IndPtr* バッファーで指定された長さは SQL_COLUMN_IGNORE で、列にはデフォルト値がありませんでした。
24000	カーソル状態が無効です。	*StatementHandle* は実行状態にありましたが、 *StatementHandle* に関連する結果セットがありませんでした。 `SQLExecute()` または `SQLExecDirect()` の後でアプリケーションは `SQLFetch()` または `SQLFetchScroll()` を呼び出しませんでした。
40001	シリアライズ障害。	トランザクションは、別のトランザクションとのリソース・デッドロックのためにロールバックされました。
40003	ステートメントの完了は不明。	関連した接続がこの関数の実行中に失敗して、トランザクションの状態を判別できません。
42000	構文エラーまたはアクセス違反。	CLI は *Operation* 引数で要求された操作の実行に必要な行をロックできませんでした。
44000	WITH CHECK OPTION 違反。	*Operation* 引数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK で、挿入または更新がビュー付き表または WITH CHECK OPTION を指定して作成したビュー付き表から派生した表で実行され、挿入または更新の影響を受ける 1 つ以上の行がビュー付き表に含まれなくなります。
HY000	一般的なエラーです。	特定の SQLSTATE が用意されておらず、しかもインプリメンテーション独自の SQLSTATE も定義されていないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリー割り振りエラー。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 15. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY011	この時点で無効な操作です。	SQLFetch() または SQLFetchScroll() の呼び出しから SQLBulkOperations の呼び出しまでの間に SQL_ATTR_ROW_STATUS_PTR ステートメント属性が設定されました。
HY013	想定外のメモリー処理エラーです。	CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	*Operation* 引数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK であり、データ値は NULL ポインターであり、列長値は 0、 SQL_DATA_AT_EXEC、SQL_COLUMN_IGNORE、 SQL_NULL_DATA のいずれでもでないか、または SQL_LEN_DATA_AT_EXEC_OFFSET 以下でした。 *Operation* 引数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK であり、データ値は NULL ポインター以外、 C データ・タイプは SQL_C_BINARY または SQL_C_CHAR、そして列長値は 0 よりも小さい値ですが、SQL_DATA_AT_EXEC、SQL_COLUMN_IGNORE、SQL_NTS、SQL_NULL_DATA のいずれでもないか、または SQL_LEN_DATA_AT_EXEC_OFFSET 以下でした。 長さ/標識バッファーの値は SQL_DATA_AT_EXEC でした。 SQL タイプは、SQL_LONGVARCHAR、 SQL_LONGVARBINARY、または長いデータ・タイプでした。また、SQLGetInfo() の情報タイプ SQL_NEED_LONG_DATA_LEN は 『Y』 でした。 *Operation* 引数は SQL_ADD で、 SQL_ATTR_USE_BOOKMARKS ステートメント属性は SQL_UB_VARIABLE に設定され、列 0 は長さがこの結果セットのブックマークの最大長に等しくないバッファーにバインドされました。 (この長さは IRD の SQL_DESC_OCTET_LENGTH フィールドに示されていて、SQLDescribeCol()、SQLColAttribute()、または SQLGetDescField() を呼び出すことによって取得できます。)

表 15. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
HY092	無効な属性 ID。	*Operation* 引数に指定された値は無効でした。 *Operation* 引数は SQL_ADD、 SQL_UPDATE_BY_BOOKMARK、または SQL_DELETE_BY_BOOKMARK で、SQL_ATTR_CONCURRENCY ステートメント属性は SQL_CONCUR_READ_ONLY に設定されました。 *Operation* 引数は SQL_DELETE_BY_BOOKMARK、SQL_FETCH_BY_BOOKMARK、または SQL_UPDATE_BY_BOOKMARK で、ブックマーク列はバインドされていないか、または SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF に設定されました。
HYC00	オプショナル・フィーチャーはインプリメントされませんでした。	CLI またはデータ・ソースは、 *Operation* 引数で要求した操作をサポートしていません。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、照会タイムアウト期間が満了しました。タイムアウト期間は、 *Attribute* 引数として SQL_ATTR_QUERY_TIMEOUT を指定した SQLSetStmtAttr() を通して設定します。
HYT01	接続タイムアウトになりました。	データ・ソースが要求に応答する前に、接続タイムアウト期間が満了しました。接続タイムアウト期間は、SQLSetConnectAttr()、SQL_ATTR_CONNECTION_TIMEOUT を使用して設定します。

制限

なし。

SQLCancel 関数 (CLI) - ステートメントの取り消し

いくつかの部分に分かれた長いデータの送信と取得のための実行時データ・シーケンスを早期終了することができます。また、これを使って、別のスレッドで呼び出された関数を取り消すこともできます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

構文

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

関数引数

表 16. SQLCancel 引数

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

使用法

SQLExecDirect() または SQLExecute() が SQL_NEED_DATA を返して実行時データ・パラメーターの値を請求した後で、 SQLCancel() を使用して、いくつかの部分に分かれた長いデータの送信と検索のための実行時データ・シーケンスを取り消すことができます。 SQLCancel() は、シーケンスの中の最後の SQLParamData() の前であれば、いつでも呼び出すことができます。このシーケンスを取り消した後で、アプリケーションは、 SQLExecute() または SQLExecDirect() を呼び出して、実行時データ・シーケンスを再開することができます。

ステートメントに対して何も処理が行われていない場合は SQLCancel() には何の効力もありません。アプリケーションがカーソルをクローズするには、SQLCancel() を呼び出すのではなく、 SQLFreeStmt() を使用する必要があります。

ホスト・データベース上での照会の取り消し

ネイティブ割り込みサポートのないサーバー (DB2 for z/OS®、バージョン 7 以前、およびIBM DB2 for IBM i など) に対して `SQLCancel()` を呼び出す場合、サーバーに対応する DCS データベース項目のカタログ時に `INTERRUPT_ENABLED` オプションが設定されていなければなりません。

`INTERRUPT_ENABLED` オプションが設定されており、`SQLCancel()` がサーバーによって受信されると、サーバーは接続を除去し、作業単位をロールバックします。アプリケーションは、サーバーへの接続が終了したことを示す、SQL30081N エラーを受け取ります。アプリケーションが追加のデータベース要求を処理するには、アプリケーションがデータベース・サーバーとの新規接続を確立する必要があります。

非同期処理の取り消し

アプリケーションが関数を非同期で呼び出した後、その関数を繰り返し呼び出して、処理を完了したかどうかを判別します。関数が処理を続けている場合、SQL_STILL_EXECUTING を戻します。

SQL_STILL_EXECUTING を戻す関数の呼び出しの後で、アプリケーションは SQLCancel() を呼び出して、関数を取り消すことができます。取り消し要求が正常に行われる場合、SQL_SUCCESS が戻されます。このメッセージは、関数が実際に取り消されたことを示すわけではありません。取り消し要求が処理されたことを示します。そのため、アプリケーションは、戻りコードが SQL_STILL_EXECUTING ではなくなるまで、元の関数の呼び出しを続ける必要があります。関数が正常に取り消された場合、その関数の戻りコードは SQL_ERROR および SQLSTATE HY008 (操作が取り消されました。) です。関数が通常の処理を完了して成功した場合、戻りコードは SQL_SUCCESS または SQL_SUCCESS_WITH_INFO です。関数が取り消し以外の理由で失敗した場合、戻りコードは SQL_ERROR および HY008 (操作が取り消されました。) 以外の SQLSTATE です。

マルチスレッド・アプリケーションでの関数の取り消し

マルチスレッド・アプリケーションでは、ステートメント上の同期的に実行中の関数を取り消すことができます。アプリケーションが関数を取り消すには、ターゲット関数で使用したのと同じステートメント・ハンドルを使って SQLCancel() を呼び出します (ただし異なるスレッド上で)。関数を取り消す方法は、オペレーティン

グ・システムによって異なります。 SQLCancel() の戻りコードは CLI が要求を正常に処理したかどうかを示すだけです。返される可能性があるのは SQL_SUCCESS または SQL_ERROR だけで、SQLSTATE は返されません。元の関数を取り消すと、SQL_ERROR および SQLSTATE HY008 (操作が取り消されました。) が返されます。

SQL ステートメントの実行中に、このステートメントの実行を取り消すために SQLCancel() が別のスレッドで呼び出されると、その実行が成功して SQL_SUCCESS が返される一方、取り消しも成功する可能性があります。この場合、CLI は、ステートメント実行によりオープンされたカーソルが取り消しによってクローズされたと想定するので、アプリケーションはそのカーソルを使用できなくなります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

注: SQL_SUCCESS とは、関数呼び出しが取り消されたことではなく、取り消し要求が処理されたことを意味します。

診断

表 17. SQLCancel SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY018	サーバーは取り消し要求を拒否しました。	サーバーが取り消し要求を拒否しました。
HY506	ファイルのクローズ・エラーです。	SQLParamData()/SQLPutData() を使用して LOB データを分割挿入しているときに、 CLI によって生成された一時ファイルをクローズする際に、エラーが発生しました。

制限

なし。

例

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

SQLCloseCursor 関数 (CLI) - カーソルのクローズとペンディング結果の廃棄

ステートメントにオープンしていたカーソルをクローズして、ペンディング中の結果を廃棄します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

構文

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

関数引数

表 18. SQLCloseCursor 引数

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

使用法

アプリケーションは、SQLCloseCursor() を呼び出した後、同じかまたは別のパラメーター値を指定して SELECT ステートメントを再実行すれば、カーソルを再オープンすることができます。 SQLCloseCursor() は、トランザクションの完了前に呼び出すことができます。

カーソルがオープンしていない場合、 SQLCloseCursor() は SQLSTATE 24000 (無効なカーソル状態) を返します。 SQLCloseCursor() 呼び出しは、 SQL_CLOSE オプションを指定した SQLFreeStmt() の呼び出しと同等ですが、相違点として、カーソルがステートメントにオープンしていない場合に、 SQLCloseCursor() から SQLSTATE 24000 (無効なカーソル状態) が返されると、 SQL_CLOSE を指定した SQLFreeStmt() はアプリケーションに対して効果はありません。

ステートメント属性 SQL_ATTR_CLOSE_BEHAVIOR を使用して、カーソルのクローズ時に、カーソルの操作中に設定された読み取りロックの解放を CLI が試みる必要があるかどうかを指示することができます。 SQL_ATTR_CLOSE_BEHAVIOR を SQL_CC_RELEASE に設定すると、データベース・マネージャーは、そのカーソルに設定されていたすべての読み取りロック (ある場合) の解放を試みます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 19. SQLCloseCursor SQLSTATE

SQLSTATE	説明	解説
01000	通常の警告	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
24000	カーソル状態が無効です。	*StatementHandle* でカーソルがオープンしていませんでした。 (これが返されるのは、CLI バージョン 5 またはそれ以降の場合だけです。)
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	*StatementHandle* で非同期実行関数が呼び出されましたが、この関数が呼び出された時点でまだ実行中でした。 SQLExecute() または SQLExecDirect() が *StatementHandle* で呼び出され、SQL_NEED_DATA を戻しました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。

制限

なし。

例

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

SQLColAttribute 関数 (CLI) - 列属性を戻す

結果セット内の列について記述子情報を返します。記述子情報は、文字ストリング、32 ビットの記述子従属値、または整数値として返されます。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLColAttributeW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

Windows 64 ビット環境では、構文は以下のようになります。

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

その他のすべてのプラットフォームの場合、構文は以下のようになります。

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

関数引数

表 20. SQLColAttribute 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*ColumnNumber*	入力	フィールド値を検索する IRD の中のレコードの番号。この引数は、結果データの列番号に対応し、その番号は 1 で始まり、左から右へ連続で順序付けられています。列は任意の順序で記述できます。 この引数内に列 0 を指定できますが、SQL_DESC_TYPE と SQL_DESC_OCTET_LENGTH を除くすべての値が、未定義の値を返すことになります。
SQLSMALLINT	*FieldIdentifier*	入力	返されることになっている IRD の行 *ColumnNumber* にあるフィールド (59 ページの表 21 を参照)。
SQLPOINTER	*CharacterAttributePtr*	出力	フィールドが文字ストリングの場合、 IRD の *ColumnNumber* 行の *FieldIdentifier* フィールド内の値を返すバッファーを指すポインター。それ以外の場合は、フィールドは未使用になります。
SQLINTEGER	*BufferLength*	入力	フィールドが文字ストリングの場合に、**CharacterAttributePtr* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。それ以外の場合、このフィールドは無視されます。

表 20. SQLColAttribute 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*StringLengthPtr*	出力	**CharacterAttributePtr* に戻すのに使える総バイト数 (文字データの場合の NULL 終止符文字のバイト・カウントを除く) を戻すバッファーを指すポインター。 文字データの場合、戻りに使用できるバイト数が *BufferLength* 以上の場合、**CharacterAttributePtr* の記述子情報は *BufferLength* から NULL 終止符文字の長さを減算した長さに切り捨てられ、CLI によってヌル終了になります。 その他のすべてのデータのタイプについては、*BufferLength* の値は無視されて、 CLI は、**CharacterAttributePtr* のサイズを 32 ビットと想定します。
SQLLEN* (Windows 64 ビット) または SQLPOINTER	*NumericAttributePtr*	出力	フィールドが SQL_DESC_COLUMN_LENGTH のような数値記述子の場合、 IRD の *ColumnNumber* 行の *FieldIdentifier* フィールド内の値を返すバッファーを指すポインター。それ以外の場合は、フィールドは未使用になります。

使用法

SQLColAttribute() は、情報を **NumericAttributePtr* または **CharacterAttributePtr* に返します。整数情報は、32 ビットの符号付き値として、**NumericAttributePtr* に返されます。その他のすべてのフォーマットの情報は、**CharacterAttributePtr* に返されます。情報が **NumericAttributePtr* に返されるとき、CLI は、*CharacterAttributePtr*、*BufferLength*、および *StringLengthPtr* を無視します。情報が **CharacterAttributePtr* に返されるとき、 CLI は、*NumericAttributePtr* を無視します。

SQLColAttribute() は、IRD の記述子フィールドからの値を返します。関数は、記述子ハンドルではなくステートメント・ハンドルを使用して呼び出されます。以下の表にリストされている、SQLColAttribute() が *FieldIdentifier* 値に応じて戻す値は、該当する IRD ハンドルを使用して SQLGetDescField() を呼び出すことにより取り出すこともできます。

現在定義されている記述子タイプ、そのタイプが (おそらく別の名前で) 導入されている CLI のバージョン、およびそのタイプに関する情報が返される引数を、以下の表に示します。さまざまなデータ・ソースを利用するために、より多くの記述子タイプが今後定義される見込みです。

CLI は、記述子タイプのおのおのについて値を返さなければなりません。記述子タイプがデータ・ソースに適用されない場合、他に断り書きがない限り、 CLI は、**StringLengthPtr* に 0 を返すか、または **CharacterAttributePtr* に空ストリングを返します。

次の表には、 SQLColAttribute() によって返される記述子タイプがリストされています。

表 21. *SQLColAttribute* 引数

FieldIdentifier	情報の戻り先	説明
SQL_DESC_AUTO_UNIQUE_VALUE (DB2 CLI v2)	Numeric AttributePtr	列データ・タイプが自動増分データ・タイプであるかどうかを示します。 DB2 SQL データ・タイプの場合はすべて、SQL_FALSE が *NumericAttributePtr* に返されます。現在、列が ID 列かどうかを CLI は確かめられないので、常に SQL_FALSE が戻されます。このような制限事項は、ODBC 仕様に全面的に準じているわけではありません。 UNIX、および Windows サーバー用の将来のバージョンの CLI では、auto-unique のサポートが設けられる予定です。
SQL_DESC_BASE_COLUMN_NAME (DB2 CLI v5)	Character AttributePtr	セット列用の基本列名。基本列名が存在しない場合 (列が式になっている場合など) は、この変数には空ストリングが入ります。 この情報は、読み取り専用フィールドである IRD の SQL_DESC_BASE_COLUMN_NAME レコード・フィールドから返されます。
SQL_DESC_BASE_TABLE_NAME (DB2 CLI v5)	Character AttributePtr	列を含む基本表の名前。基本表名が定義できないか適用外である場合、この変数には空ストリングが入ります。
SQL_DESC_CASE_SENSITIVE (DB2 CLI v2)	Numeric AttributePtr	列データ・タイプが大文字小文字の区別があるタイプであるかどうかを示します。 SQL_TRUE または SQL_FALSE のどちらが *NumericAttributePtr* に返されるかは、データ・タイプに依存します。 大文字小文字の区別は GRAPHIC データ・タイプには適用されず、SQL_FALSE が返されます。 非文字データ・タイプと XML データ・タイプには SQL_FALSE が返されます。
SQL_DESC_CATALOG_NAME (DB2 CLI v2)	Character AttributePtr	CLI は 1 つの表につき 2 つの部分からなる命名しかサポートしないため、空ストリングが返されます。
SQL_DESC_CONCISE_TYPE (DB2 CLI v5)	Numeric AttributePtr	コンサイス・データ・タイプ 日時データ・タイプの場合、このフィールドはコンサイス・データ・タイプ、例えば、SQL_TYPE_TIME を返します。 この情報は、IRD の SQL_DESC_CONCISE_TYPE レコード・フィールドから返されます。
SQL_DESC_COUNT (DB2 CLI v2)	Numeric AttributePtr	結果セット内の列数が、*NumericAttributePtr* に返されます。
SQL_DESC_DISPLAY_SIZE (DB2 CLI v2)	Numeric AttributePtr	文字フォーマットでデータを表示するのに必要な最大バイト数が、 *NumericAttributePtr* に返されます。 おのおのの列タイプの表示サイズについては、『データ・タイプ表示サイズ』の表を参照してください。

表 21. SQLColAttribute 引数 (続き)

FieldIdentifier	情報の戻り先	説明
SQL_DESC_DISTINCT_TYPE (DB2 CLI v2)	Character AttributePtr	列のユーザー定義特殊タイプ名が、*CharacterAttributePtr* に返されます。列が組み込み SQL タイプであってユーザー定義特殊タイプ名ではない場合、空ストリングが返されます。 **注:** これは、ODBC によって定義された記述子属性のリストに対する IBM 定義の拡張機能です。
SQL_DESC_FIXED_PREC_SCALE (DB2 CLI v2)	Numeric AttributePtr	SQL_TRUE は、列がデータ・ソース固有の固定精度および非ゼロのスケールを持っている場合です。 SQL_FALSE は、列がデータ・ソース固有の固定精度および非ゼロのスケールを持っていない場合です。 DB2 SQL データ・タイプの場合はすべて、SQL_FALSE が *NumericAttributePtr* に返されます。
SQL_DESC_LABEL (DB2 CLI v2)	Character AttributePtr	列ラベルが、*CharacterAttributePtr* に返されます。列にラベルがない場合、列名または列式が返されます。列にラベルがなく、名前もない場合は、空ストリングが返されます。
SQL_DESC_LENGTH (DB2 CLI v2)	Numeric AttributePtr	文字ストリングまたはバイナリー・データ・タイプの長さを示すエレメント数 (SQLCHAR または SQLWCHAR) の最大値または実際の値。これは、固定長データ・タイプの場合は最大エレメント長、可変長データ・タイプの場合は実際のエレメント長となります。その値からは常に、文字ストリングの終わりを示すヌル終了バイトが除かれています。 この情報は、IRD の SQL_DESC_LENGTH レコード・フィールドから返されます。 XML データ・タイプの場合、この値は 0 です。
SQL_DESC_LITERAL_PREFIX (DB2 CLI v5)	Character AttributePtr	この VARCHAR(128) レコード・フィールドには、CLI がこのデータ・タイプのリテラル用の接頭部として認識する文字 (複数を含む) が入っています。リテラルの接頭部が適用外であるデータ・タイプに対しては、このフィールドに空ストリングが入れられます。
SQL_DESC_LITERAL_SUFFIX (DB2 CLI v5)	Character AttributePtr	この VARCHAR(128) レコード・フィールドには、CLI がこのデータ・タイプのリテラル用の接尾部として認識する文字 (複数を含む) が入っています。リテラルの接尾部が適用外であるデータ・タイプに対しては、このフィールドに空ストリングが入れられます。
SQL_DESC_LOCAL_TYPE_NAME (DB2 CLI v5)	Character AttributePtr	この VARCHAR(128) レコード・フィールドには、データ・タイプの正規名とは異なる、データ・タイプ用のローカライズされた (ネイティブ言語の) 名前が入ります。ローカライズされた名前がない場合は、空ストリングが返されます。このフィールドは、表示の目的においてのみ使用されます。ストリングの文字セットはロケールに依存しており、通常はサーバーのデフォルト文字セットです。

表 21. *SQLColAttribute* 引数 (続き)

FieldIdentifier	情報の戻り先	説明
SQL_DESC_NAME (DB2 CLI v2)	Character AttributePtr	列 *ColumnNumber* の名前が、*CharacterAttributePtr* に返されます。列が式である場合は、列番号が返されます。 いずれの場合にも、SQL_DESC_UNNAMED が SQL_NAMED に設定されます。列名または列別名がない場合は、空ストリングが返されて、SQL_DESC_UNNAMED が SQL_UNNAMED に設定されます。 この情報は、IRD の SQL_DESC_NAME レコード・フィールドから返されます。
SQL_DESC_NULLABLE (DB2 CLI v2)	Numeric AttributePtr	*ColumnNumber* によって識別される列に NULL を入れることができる場合、 SQL_NULLABLE が *NumericAttributePtr* に返されます。 列が NULL を受け入れないように制約されている場合、 SQL_NO_NULLS が *NumericAttributePtr* に返されます。 この情報は、 IRD の SQL_DESC_NULLABLE レコード・フィールドから返されます。
SQL_DESC_NUM_PREX_RADIX (DB2 CLI v5)	Numeric AttributePtr	• SQL_DESC_TYPE フィールド内のデータ・タイプが近似的なデータ・タイプである場合、この SQLINTEGER フィールドには 2 の値が入ります。SQL_DESC_PRECISION フィールドにビット数が入っているからです。 • SQL_DESC_TYPE フィールド内のデータ・タイプが厳密な数データ・タイプである場合は、このフィールドの値は 10 になります。SQL_DESC_PRECISION フィールドは小数桁数を含むからです。

表 21. SQLColAttribute 引数 (続き)

FieldIdentifier	情報の戻り先	説明
SQL_DESC_OCTET_LENGTH (DB2 CLI v2)	Numeric AttributePtr	列に関連したデータのバイト数が、*NumericAttributePtr* に返されます。これは、SQL_C_DEFAULT が C データ・タイプに指定されているときにこの列のフェッチ時または SQLGetData() 実行時に転送されたデータの長さをバイト数で表したものです。個々の SQL データ・タイプの長さについては、『データ・タイプ長』の表を参照してください。 *ColumnNumber* 内で識別される列が、固定長の文字ストリングまたはバイナリー・ストリング (例えば、SQL_CHAR または SQL_BINARY) である場合、実際の長さが返されます。 *ColumnNumber* 内で識別される列が、可変長の文字ストリングまたはバイナリー・ストリング (例えば、SQL_VARCHAR または SQL_BLOB) である場合、最大長が返されます。 *ColumnNumber* で識別される列のタイプが SQL_XML の場合、0 が戻されます。
SQL_DESC_PRECISION (DB2 CLI v2)	Numeric AttributePtr	列が SQL_DECIMAL、 SQL_NUMERIC、SQL_DOUBLE、 SQL_FLOAT、 SQL_INTEGER、SQL_REAL、または SQL_SMALLINT の場合、数値の精度 (有効桁数) が、*NumericAttributePtr* に返されます。 列が文字 SQL データ・タイプである場合、*NumericAttributePtr* に返される精度は、列に入れることのできる SQLCHAR または SQLWCHAR エレメントの最大数を示します。 列が GRAPHIC SQL データ・タイプの場合、*NumericAttributePtr* に返される精度は、列に入れることのできる 2 バイト・エレメントの最大数を指定します。 列が XML データ・タイプの場合、精度は 0 です。 個々の SQL データ・タイプの精度については、『データ・タイプ精度』の表を参照してください。 この情報は、IRD の SQL_DESC_PRECISION レコード・フィールドから返されます。
SQL_DESC_SCALE (DB2 CLI v2)	Numeric AttributePtr	列のスケールの属性が返されます。個々の SQL データ・タイプのスケールについては、『データ・タイプ・スケール』の表を参照してください。 この情報は、IRD の SCALE レコード・フィールドから返されます。

表 21. *SQLColAttribute* 引数 (続き)

FieldIdentifier	情報の戻り先	説明
SQL_DESC_SCHEMA_NAME (DB2 CLI v2)	Character AttributePtr	列を含む表のスキーマが、*CharacterAttributePtr* に返されます。表を含むスキーマの名前が返されます。スキーマの名前が 8 文字未満の場合は、余分な文字としてスペースが追加されます。
SQL_DESC_SEARCHABLE (DB2 CLI v2)	Numeric AttributePtr	列データ・タイプが検索可能であるかどうかを示します。 • SQL_PRED_NONE (DB2 CLI v2 での SQL_UNSEARCHABLE): 列を WHERE 文節に使用できない場合。 • SQL_PRED_CHAR (DB2 CLI v2 での SQL_LIKE_ONLY): LIKE 述部を用いてのみ、列を WHERE 文節に使用できる場合。 • SQL_PRED_BASIC (DB2 CLI v2 での SQL_ALL_EXCEPT_LIKE): LIKE を除くすべての比較演算子を用いて、列を WHERE 文節に使用できる場合。 • SQL_SEARCHABLE: どの比較演算子を指定したときでも WHERE 文節で列を使用できる場合。
SQL_DESC_TABLE_NAME (DB2 CLI v2)	Character AttributePtr	列を含む表名が返されます。表名が定義できないか適用外である場合、この変数には空ストリングが入ります。
SQL_DESC_TYPE (DB2 CLI v2)	Numeric AttributePtr	*ColumnNumber* で識別される列の SQL データ・タイプが、 *NumericAttributePtr* に返されます。返される可能性のある値は、「CLI 用の記号およびデフォルトのデータ・タイプ」表にリストされています。 *ColumnNumber* が 0 に等しければ、可変長ブックマークの場合は SQL_BINARY が返され、固定長ブックマークの場合は SQL_INTEGER が返されます。 日時データ・タイプの場合、このフィールドは冗長データ・タイプ、例えば SQL_DATETIME を返します。 この情報は、IRD の SQL_DESC_TYPE レコード・フィールドから返されます。
SQL_DESC_TYPE_NAME (DB2 CLI v2)	Character AttributePtr	列のタイプ (SQL ステートメントに入力したとおりのもの) が、 *CharacterAttributePtr* に返されます。 各データ・タイプの詳細は、『CLI 用の記号およびデフォルトのデータ・タイプ』を参照してください。
SQL_DESC_UNNAMED (DB2 CLI v5)	Numeric AttributePtr	SQL_NAMED または SQL_UNNAMED。 IRD の SQL_DESC_NAME フィールドに列別名または列名が入っている場合、 SQL_NAMED が返されます。列名も列別名も入っていない場合は、SQL_UNNAMED が返されます。 この情報は、IRD の SQL_DESC_UNNAMED レコード・フィールドから返されます。

表 21. SQLColAttribute 引数 (続き)

FieldIdentifier	情報の戻り先	説明
SQL_DESC_UNSIGNED (DB2 CLI v2)	Numeric AttributePtr	列データ・タイプが無符号タイプであるかどうかを示します。 すべての非数値データ・タイプについては、SQL_TRUE が *NumericAttributePtr* に返され、すべての数値データ・タイプについては、SQL_FALSE が返されます。
SQL_DESC_UPDATABLE (DB2 CLI v2)	Numeric AttributePtr	列のデータ・タイプが更新可能なデータ・タイプであるかどうかを指定します。 • 結果セット列が読み取り専用の場合、SQL_ATTR_READONLY が返されます。 • 結果セット列が読み取り/書き込み指定の場合、SQL_ATTR_WRITE が返されます。 • 結果セット列が更新可能かどうかが不明である場合、SQL_ATTR_READWRITE_UNKNOWN が返されます。

この関数は、SQLDescribeCol() の代替として拡張性のあるものです。SQLDescribeCol() は、ANSI-89 SQL に基づく記述子情報の固定セットを返します。SQLColAttribute() は、ANSI SQL-92 および DBMS ベンダーの拡張機能で使用できる、記述子情報のより広範なセットにアクセスできるようになっています。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 22. SQLColAttribute SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	バッファー **CharacterAttributePtr* は、ストリング値を全部返すのに十分な大きさではなかったので、ストリング値が切り捨てられました。 **StringLengthPtr* には、切り捨て前のストリング値の長さが戻されます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
07005	ステートメントが結果セットを返しませんでした。	*StatementHandle* に関連したステートメントが結果セットを返しませんでした。記述する列がありませんでした。
07009	記述子索引が無効です。	*ColumnNumber* に指定された値が 0 と同等であり、SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF でした。引数 *ColumnNumber* に指定された値は、結果セット内の列数より大きい値でした。

表 22. SQLColAttribute SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から *MessageText バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、 *StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	SQLPrepare() または SQLExecDirect() を *StatementHandle* 用に呼び出す前に、この関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* に指定された値は、0 より小さい値でした。
HY091	記述子フィールド ID が無効です。	引数 *FieldIdentifier* に指定された値は、定義されている値の 1 つではなく、インプリメンテーション定義の値でもありませんでした。
HYC00	ドライバーが使用できません。	引数 *FieldIdentifier* に指定された値は、CLI でサポートされていませんでした。

StatementHandle に関連した SQL ステートメントをデータ・ソースが評価する時期に応じて、 SQLPrepare() の後から SQLExecute() の前までの間に呼び出された SQLColAttribute() は、 SQLPrepare() または SQLExecute() によって返される可能性のある任意の SQLSTATE を返すことができます。

パフォーマンス上の理由から、アプリケーションは、ステートメントの実行前に SQLColAttribute() を呼び出さないようにしなければなりません。

制限

なし。

例

```
/* get display size for column */
cliRC = SQLColAttribute(hstmt,
                        (SQLSMALLINT)(i + 1),
```

```
                                SQL_DESC_DISPLAY_SIZE,
                                NULL,
                                0,
                                NULL,
                                &colDataDisplaySize)
```

SQLColAttributes 関数 (CLI) - 列属性の取得

ODBC 3.0 では SQLColAttributes() は使用すべきでない関数なので、代わりに SQLColAttribute() を使用します。

このバージョンの CLI でも引き続き SQLColAttributes() をサポートしていますが、最新の標準に準拠するように、SQLColAttribute() を CLI プログラムで使用します。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLColAttributesW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

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

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

SQLColumnPrivileges 関数 (CLI) - 表の列に関連した特権の取得

指定された表の列とそれに関連した特権のリストを返します。

この情報は SQL 結果セットで返されます。この結果セットは、照会によって生成される結果セットを処理するために使用するのと同じ関数を使用して取得することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

SQLColumnPrivileges() は、指定された表の列とそれに関連した特権のリストを返します。この情報は SQL 結果セットにして戻されます。これは、照会から生成された結果セットの処理で使用するものと同じ関数を用いて取り出すことができます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLColumnPrivilegesW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 23. SQLColumnPrivileges 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	表名のスキーマ修飾子。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名。
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ColumnName*	入力	列名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength4*	入力	*ColumnName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ColumnName* がヌル終了ストリングの場合は SQL_NTS。

使用法

結果は、SQLColumnPrivileges で戻される列にリストされている列を含む標準結果セットとして返されます。結果セットは、 TABLE_CAT、 TABLE_SCHEM、 TABLE_NAME、 COLUMN_NAME、および PRIVILEGE の順序になります。複数の特権が指定列に関連付けられている場合、各特権は個別の行として返されます。

通常のアプリケーションでは、SQLColumns() を呼び出して列特権情報を判別してから、この関数を呼び出すこともできます。アプリケーションは、SQLColumns() 結果セットの TABLE_CAT、TABLE_SCHEM、TABLE_NAME、COLUMN_NAME 列に返される文字ストリングを、この関数の入力引数として使用する必要があります。

多くの場合に SQLColumnPrivileges() の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、この呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

ColumnName 入力引数は検索パターンを受け入れますが、他のどの入力引数もこれを受け入れません。

アプリケーションが関数を呼び出して、結果セットの戻りを制限する試行がなされない場合があります。長い検索時間を短縮するため、構成キーワード SchemaList を CLI 初期設定ファイルに指定できます。そうすれば、アプリケーションが SchemaName に NULL ポインターを提供した場合に結果セットを限定できます。アプリケーションが SchemaName ストリングを指定した場合にも、出力を限定するにはやはり SchemaList キーワードを使います。したがって、指定されたスキーマ名が SchemaList ストリングでないと、空の結果セットが生成されます。

SchemaName に値として *ALL または *USRLIBL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。*ALL を指定すると、CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。 IBM DB2 for IBM i サーバーで *USRLIBL を指定すると、CLI はサーバー・ジョブの現行ライブラリーで検索します。他の DB2 サーバーでは、*USRLIBL は特別な意味を持たず、CLI はパターンとして *USRLIBL を使用して検索します。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL または *USRLIBL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLColumnPrivileges で戻される列

列 1 TABLE_CAT (VARCHAR(128) データ・タイプ)
カタログの名前。この表にカタログがない場合、この値は NULL になります。

列 2 TABLE_SCHEM (VARCHAR(128))
TABLE_NAME の入ったスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表またはビューの名前。

列 4 COLUMN_NAME (VARCHAR(128) 非 NULL)
指定された表またはビューの列の名前。

列 5 GRANTOR (VARCHAR(128))
特権を付与したユーザーの許可 ID。

列 6 GRANTEE (VARCHAR(128))
特権が付与されたユーザーの許可 ID。

列 7 PRIVILEGE (VARCHAR(128))
列の特権。これには、以下の種類があります。
- INSERT
- REFERENCES
- SELECT
- UPDATE

注: いくつかの IBM RDBMS は、列レベルでの列レベル特権を提供していません。 DB2 Database for Linux, UNIX, and Windows、DB2 for z/OS、および DB2 Server for VM and VSE は UPDATE 列特権をサポートしています。この結果セットには各更新可能な列につき 1 行が割り当てられています。DB2 Database for Linux, UNIX, and Windows、DB2 for z/OS、および DB2 Server for VM and VSE のその他すべての特権、およびその他の IBM RDBMS についてのすべての特権には、表レベルで特権が付与されている場合は、この結果セットで 1 行が割り当てられます。

列 8 IS_GRANTABLE (VARCHAR(3) データ・タイプ)
特権を付与されたユーザーが他のユーザーに特権を付与できるかどうかを示します。

「YES」または「NO」。

注: CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の `SQLColumnPrivileges()` 結果セットで定義されているものと同じです。

ある列に関連した特権が複数ある場合、各特権は結果セット内の個別の行として返されます。

戻りコード
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 24. SQLColumnPrivileges SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40001	シリアライズ障害	トランザクションは、別のトランザクションとのリソース・デッドロックのためにロールバックされました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。

表 24. SQLColumnPrivileges SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	*TableName* が NULL です。
HY010	関数のシーケンス・エラー。	非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 SQLExecute()、SQLExecDirect()、または SQLSetPos() が *StatementHandle* で呼び出され、SQL_NEED_DATA を戻しました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

例

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

SQLColumns 関数 (CLI) - 表の列の情報の取得

SQLColumns() 関数は、指定された表の中の列のリストを返します。この情報は SQL 結果セットで返されます。この結果セットは、照会によって生成される結果セットをフェッチするために使用するのと同じ関数を使用して取得することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLColumnsW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

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

関数引数

表 25. SQLColumns 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名で結果セットを修飾するための *パターン値* が入れられるバッファー。

表 25. SQLColumns 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ColumnName*	入力	列名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength4*	入力	*ColumnName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ColumnName* がヌル終了ストリングの場合は SQL_NTS。

使用法

表や表集合の列に関する情報を検索するには、この関数を使用します。アプリケーションでは、 `SQLTables()` を呼び出して表の列を判別してから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、`SQLTables()` 結果セットの TABLE_SCHEMA 列と TABLE_NAME 列に返される文字ストリングを、この関数の入力として使用しなければなりません。

`SQLColumns()` 関数は、TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および ORDINAL_POSITION の順序になっている標準の結果セットを返します。SQLColumns で戻される列は、結果セット内の列をリストしています。

SchemaName、*TableName*、および *ColumnName* 入力引数は、検索パターンを受け入れます。

アプリケーションが関数を呼び出して、結果セットの戻りを制限する試行がなされない場合があります。例えば大量の表、ビュー、および別名を含むデータ・ソースの場合、このシナリオの結果セットは非常に大きくなり、検索時間が非常に長くかかります。長い検索時間を短縮するため、構成キーワード **SchemaList** を CLI 初期設定ファイルに指定できます。そうすれば、アプリケーションが **SchemaName** に NULL ポインターを提供した場合に結果セットを限定できます。アプリケーションが **SchemaName** ストリングを指定した場合にも、出力を限定するにはやはり **SchemaList** キーワードを使います。したがって、指定されたスキーマ名が **SchemaList** ストリングでないと、空の結果セットが生成されます。

この関数は、結果セットの列に関する情報を返しません。`SQLDescribeCol()` 関数か `SQLColAttribute()` 関数を代わりに使用する必要があります。

`SQLSetConnectAttr()` の呼び出しによってか、または CLI 初期設定ファイル内の LONGDATACOMPAT キーワードの設定によって SQL_ATTR_LONGDATA_COMPAT 属性が SQL_LD_COMPAT_YES に設定された場合、LOB データ・タイプは、SQL_LONGVARCHAR、SQL_LONGVARBINARY、または SQL_LONGVARGRAPHIC と報告されます。

多くの場合に SQLColumns() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

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、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SchemaName に値として *ALL または *USRLIBL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。*ALL を指定すると、CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。 IBM DB2 for IBM i サーバーで *USRLIBL を指定すると、CLI はサーバー・ジョブの現行ライブラリーで検索します。他の DB2 サーバーでは、*USRLIBL は特別な意味を持たず、CLI はパターンとして *USRLIBL を使用して検索します。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL または *USRLIBL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLColumns で戻される列

列 1 TABLE_CAT (VARCHAR(128))
カタログの名前。この表にカタログがない場合、この値は NULL になります。

列 2 TABLE_SCHEM (VARCHAR(128))
TABLE_NAME を含むスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表、ビュー、別名、またはシノニムの名前。

列 4??COLUMN_NAME (VARCHAR(128) 非 NULL)
列の ID。指定された表、ビュー、別名、またはシノニムの列の名前。

列 5 DATA_TYPE (SMALLINT 非 NULL)
COLUMN_NAME によって識別される列の SQL データ・タイプ。DATA_TYPE は、CLI 用の記号データ・タイプおよびデフォルト・データ・タイプの表の「記号 SQL データ・タイプ」列にある値の 1 つです。

列 6 TYPE_NAME (VARCHAR(128) 非 NULL)
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。

列 7 COLUMN_SIZE (INTEGER)
DATA_TYPE 列の値が文字ストリングまたはバイナリー・ストリングを示す場合、この列には列の SQLCHAR または SQLWCHAR エレメント数で表記した最大長が入れられます。

日付、時刻、およびタイム・スタンプのデータ・タイプの場合、COLUMN_SIZE は文字データ・タイプに変換された場合に値を表示するために必要な SQLCHAR エレメントまたは SQLWCHAR エレメントの数の合計です。

数値データ・タイプの場合、COLUMN_SIZE は結果セット内の NUM_PREC_RADIX 列の値に基づいて、列に許可されている総桁数または合計ビット数のいずれかです。

XML データ・タイプの場合、長さゼロが戻されます。

データ・タイプ精度の表を参照してください。

列 8 BUFFER_LENGTH (INTEGER)
SQL_C_DEFAULT が SQLBindCol()、SQLGetData() および SQLBindParameter() 呼び出しで指定された場合に、この列からのデータを保管するための関連する C バッファーの最大バイト。この長さには、NULL 終止符文字は含まれていません。厳密な数データ・タイプを出すには、長さとして小数部や符号も考慮されます。

データ・タイプ長の表を参照してください。

列 9 DECIMAL_DIGITS (SMALLINT)
列のスケール。スケールが適用できないデータ・タイプの場合は NULL が戻されます。

データ・タイプ・スケールの表を参照してください。

列 10 NUM_PREC_RADIX (SMALLINT)
10、2、または NULL のいずれか。DATA_TYPE が近似値データ・タイプである場合、この列には値 2 が入り、COLUMN_SIZE 列にはその列で許可されているビット数が入ります。

DATA_TYPE が厳密な数データ・タイプの場合、この列には値 10 が入り、COLUMN_SIZE にはその列に許可されている小数桁数が入ります。

数値データ・タイプの場合、DBMS は 10 または 2 の NUM_PREC_RADIX を戻すことができます。

基数が適用できないデータ・タイプの場合は NULL が戻されます。

列 11 NULLABLE (SMALLINT 非 NULL)
列が NULL を受け入れない場合は SQL_NO_NULLS。

列が NULL 値を受け入れる場合は SQL_NULLABLE。

列 12 REMARKS (VARCHAR(254))
列に関する記述情報を入れることができます。この列には、情報が戻されない場合があります。詳しくは、SQL 列のキーワードと属性の最適化を参照してください。

列 13 COLUMN_DEF (VARCHAR(254))
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で囲まれていない数値リテラルの文字表示が含まれています。デフォルト値が文字ストリングの場合、この列は単一引用符で囲まれたストリングです。デフォルト値が DATE、TIME、および TIMESTAMP 列の場合などの *疑似リテラル* の場合、この列には引用符で囲まれていない疑似リテラル (CURRENT DATE など) のキーワードが入ります。

NULL をデフォルト値として指定した場合、この列は引用符で囲まれていない語 NULL を戻します。切り捨てを行わないとデフォルト値を表すことができない場合、この列には単一引用符で囲まれていない TRUNCATED が入ります。デフォルト値を指定しなかった場合、この列は NULL です。

この列には、情報が戻されない場合があります。詳しくは、SQL 列のキーワードと属性の最適化を参照してください。

列 14 SQL_DATA_TYPE (SMALLINT 非 NULL)
IRD の SQL_DESC_TYPE レコード・フィールドに現れる SQL データ・タイプ。この列は、日付、時刻、およびタイム・スタンプのデータ・タイプに関しては、SQLColumns で戻される列内の DATA_TYPE 列と同じです。

列 15 SQL_DATETIME_SUB (SMALLINT)
日時データ・タイプのサブタイプ・コード。
- SQL_CODE_DATE
- SQL_CODE_TIME
- SQL_CODE_TIMESTAMP

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。

列 16 CHAR_OCTET_LENGTH (INTEGER)
1 バイト文字セットの場合、これは COLUMN_SIZE と同じです。 XML タイプの場合、ゼロが戻されます。他のすべてのデータ・タイプの場合は、NULL が戻されます。

列 17 ORDINAL_POSITION (INTEGER 非 NULL)
表中の列の順序を示す位置。表の最初の列は 1 です。

列 18 IS_NULLABLE (VARCHAR(254))
列が NULL 可能でないことがわかっている場合はストリング「NO」が含まれており、列が NULL 可能である場合は「YES」が含まれています。

注: この結果セットは X/Open CLI の `Columns()` 結果セット仕様と同じであり、ODBC V2 に指定されている `SQLColumns()` 結果セットの拡張版です。ODBC `SQLColumns()` 結果セットには、同じ位置にあるすべての列が含まれています。

SQL 列のキーワードと属性の最適化

以下のどちらかを使用して、`SQLColumns()` 関数への呼び出しを最適化するよう CLI/ODBC ドライバーを設定できます。
- OPTIMIZESQLCOLUMNS CLI/ODBC 構成キーワード
- SQLSetConnectAttr() の SQL_ATTR_OPTIMIZESQLCOLUMNS 接続属性

上記のどちらかの値を設定すると、以下の列に入っている情報は戻されません。
- 列 12 REMARKS
- 列 13 COLUMN_DEF

戻りコード
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 26. SQLColumns SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

SQLColumns() 関数は、別名の別名からデータを戻すことはサポートしません。別名の別名に対して呼び出された場合には、SQLColumns() 関数は空の結果セットを戻します。

例

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

```
tbNamePattern,
SQL_NTS,
colNamePattern,
SQL_NTS);
```

SQLConnect 関数 (CLI) - データ・ソースへの接続

ターゲット・データベースに対する接続またはトラステッド接続を確立します。

アプリケーションは、ターゲット SQL データベースと、必要があれば許可名と認証ストリングを指定する必要があります。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

SQLAllocHandle() を使用してステートメント・ハンドルを割り振る場合、事前に接続を確立しておかなければなりません。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLConnectW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 27. SQLConnect 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル
SQLCHAR *	*ServerName*	入力	データ・ソース: データベースの名前または別名。
SQLSMALLINT	*ServerNameLength*	入力	*ServerName* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLCHAR *	*UserName*	入力	許可名 (ユーザー ID)
SQLSMALLINT	*UserNameLength*	入力	*UserName* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLCHAR *	*Authentication*	入力	認証ストリング (パスワード)

表 27. SQLConnect 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*AuthenticationLength*	入力	*Authentication* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。

使用法

IBM RDBMS のターゲット・データベース (データ・ソース とも呼ぶ) は、データベース別名です。アプリケーションは、SQLDataSources() を呼び出して、接続できるデータベースのリストを取得することができます。

SQLConnect() の入力長さ引数 (*ServerNameLength*、 *UserNameLength*、*AuthenticationLength*) は、その関連データの実際のエレメント数 (SQLCHAR または SQLWCHAR) による長さ (NULL 終止符文字を含まない) に設定するか、または SQL_NTS に設定して関連データがヌル終了ストリングであることを指定できます。

ServerName および *UserName* 引数値にはブランクを入れてはなりません。

CLI を使用して書かれたストアード・プロシージャーで、*NULL* の SQLConnect() 呼び出しを行う必要があります。 NULL SQLConnect() とは、*ServerName*、*UserName*、および *Authentication* 引数ポインターがすべて NULL に設定され、それらの長さ引数がすべて 0 に設定されているものです。NULL の SQLConnect() の場合でも、まず SQLAllocHandle() を呼び出す必要がありますが、SQLDisconnect() より先に SQLEndTran() を呼び出す必要はありません。

トラステッド接続を作成するには、SQLConnect() を呼び出す前に、接続属性 SQL_ATTR_USE_TRUSTED_CONTEXT を指定します。データベース・サーバーが接続を信頼できるものとして受け入れる場合、その接続はトラステッド接続と見なされます。そうでない場合、接続は通常の接続になり、警告が戻されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 28. SQLConnect SQLSTATE

SQLSTATE	説明	解説
01679	トラステッド接続を確立できません。	CLI はトラステッド接続を要求しましたが、接続の trust 属性が、データベース・サーバー上の信頼できるコンテキスト・オブジェクトと一致しません。接続は許可されますが、トラステッド接続ではなく、通常の接続になります。
08001	データ・ソースに接続できませんでした。	CLI はデータ・ソース (サーバー) との接続を確立できませんでした。 組み込み SQL を経由して確立された接続がすでにあるため、接続要求が拒否されました。

表 28. SQLConnect SQLSTATE (続き)

SQLSTATE	説明	解説
08002	接続が使用中です。	指定された *ConnectionHandle* はデータ・ソースとの接続を確立するためにすでに使用されており、接続がまだオープンしています。
08004	アプリケーション・サーバーが、接続の確立を拒否しました。	データ・ソース (サーバー) は、接続の確立を拒否しました。
28000	許可指定が無効。	引数 *UserName* で指定された値または引数 *Authentication* で指定された値が、データ・ソースで定義されている制限に違反しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *ServerNameLength* に指定された値は 0 より小さい値でしたが、 SQL_NTS に等しくなく、引数 *ServerName* は NULL ポインターではありませんでした。 引数 *UserNameLength* に指定された値は 0 より小さい値でしたが、 SQL_NTS に等しくなく、引数 *UserName* は NULL ポインターではありませんでした。 引数 *AuthenticationLength* に指定された値は 0 より小さい値でしたが、 SQL_NTS に等しくなく、引数 *Authentication* は NULL ポインターではありませんでした。
HY501	データ・ソース名が無効です。	引数 *ServerName* 内に、無効なデータ・ソース名を指定しました。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

IBM RDBMS の暗黙接続 (またはデフォルト・データベース) オプションは、サポートされません。 SQLConnect() を呼び出さないと、SQL ステートメントを実行できません。

例

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

SQLCopyDesc 関数 (CLI) - ハンドル間での記述子情報のコピー

1 つの記述子ハンドルからもう 1 つの記述子ハンドルへと記述子情報をコピーします。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

構文

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

関数引数

表 29. SQLCopyDesc 引数

データ・タイプ	引数	使用法	説明
SQLHDESC	*SourceDescHandle*	入力	ソース記述子ハンドル
SQLHDESC	*TargetDescHandle*	入力	ターゲット記述子ハンドル。 *TargetDescHandle* は、アプリケーション記述子または IPD に対するハンドルになり得ます。 SQLCopyDesc() は、*TargetDescHandle* が IRD に対するハンドルである場合に、 SQLSTATE HY016 を返します (インプリメンテーション記述子を変更することはできません)。

使用法

SQLCopyDesc() を呼び出して、ソース記述子ハンドルのフィールドをターゲット記述子ハンドルにコピーします。フィールドは、アプリケーション記述子または IPD にはコピーできますが、 IRD にはコピーできません。フィールドは、アプリケーション記述子からでもインプリメンテーション記述子からでもコピーできます。

記述子のすべてのフィールドは、 SQL_DESC_ALLOC_TYPE (これは記述子ハンドルが自動的に割り振られたか明示的に割り振られたかを指定します) を除いて、そのフィールドが宛先記述子用に定義されていてもいなくても、コピーされます。フィールドをコピーすると、*TargetDescHandle* 内の既存のフィールドは上書きされます。

SourceDescHandle と *TargetDescHandle* が 2 つの別々の接続または環境内にある場合でも、すべての記述子フィールドがコピーされます。

SQLCopyDesc() の呼び出しは、エラーが発生した場合は、直ちに打ち切られます。

SQL_DESC_DATA_PTR フィールドがコピーされるとき、整合性チェックが行われます。整合性チェックに失敗した場合、SQLSTATE HY021 (記述子情報が矛盾します。) が返されて、 SQLCopyDesc() の呼び出しは直ちに打ち切られます。

注: 記述子ハンドルは、接続または環境をまたがってコピーできます。しかし、アプリケーションは、SQLCopyDesc() を呼び出すよりは、明示的に割り振られた記述

子ハンドルを *StatementHandle* に関連付けて、 1 つの記述子からもう 1 つの記述子へとフィールドをコピーできるようにします。明示的に割り振られた記述子を、同じ *ConnectionHandle* 上の別の *StatementHandle* に関連付けることができます。それには、SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC ステートメント属性を、明示的に割り振られた記述子のハンドルに設定します。これが行われると、 1 つの記述子から別の記述子へ記述子フィールドの値をコピーするために SQLCopyDesc() を呼び出す必要はなくなります。

記述子ハンドルは、もう 1 つ別の *ConnectionHandle* 上の *StatementHandle* には関連付けできませんが、異なる *ConnectionHandle* 上の *StatementHandle* で同じ記述子フィールド値を使用するには、 SQLCopyDesc() の呼び出しが必要になります。

表の間での行のコピー

1 つのステートメント・ハンドル上の ARD は、別のステートメント・ハンドル上の APD として機能できます。このことから、アプリケーション・レベルでのデータのコピーをしなくても、表間で行のコピーを行うことができます。これを行うには、アプリケーションが SQLCopyDesc() を呼び出して、表からフェッチした行を記述する ARD のフィールドを、別のステートメント・ハンドル上の INSERT ステートメントのパラメーター用の APD にコピーします。 SQLGetInfo() の呼び出しに対してドライバーにより返される SQL_ACTIVE_STATEMENTS の *InfoType* は、後続の操作のために 1 より大きい値にする必要があります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

SQLCopyDesc() が SQL_ERROR または SQL_SUCCESS_WITH_INFO を返した場合、 SQL_HANDLE_DESC の HandleType および *TargetDescHandle* で `Handle` を用いて `SQLGetDiagRec()` を呼び出せば、関連した SQLSTATE 値を入手することができます。呼び出しで無効な *SourceDescHandle* が渡された場合は、SQL_INVALID_HANDLE が返されますが、SQLSTATE は返されません。

エラーが返されたとき、SQLCopyDesc() の呼び出しは直ちに打ち切られ、*TargetDescHandle* 記述子内のフィールドの内容は未定義になります。

表 30. SQLCopyDesc SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、 CLI と接続を試行していたデータ・ソースとの間の通信リンクが失敗しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。

表 30. SQLCopyDesc SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY007	関連ステートメントが準備されていません。	*SourceDescHandle* は IRD と関連付けられており、関連付けられているステートメント・ハンドルが準備済みまたは実行済みの状態にはありません。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。
HY016	インプリメンテーション行の記述子を修正できません。	*TargetDescHandle* が IRD に関連付けられました。
HY021	記述子情報が矛盾します。	整合性チェック時にチェックされた記述子情報は、整合性がとれていませんでした。
HY092	オプション・タイプが範囲外です。	SQLCopyDesc() の呼び出しにより、 SQLSetDescField() を呼び出すプロンプトが出されましたが、 **ValuePtr* は、 *TargetDescHandle* 上の *FieldIdentifier* 引数に対して有効ではありませんでした。

制限

なし。

例

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

/* ... */

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

SQLCreateDb 関数 (CLI) - データベースの作成

SQLCreateDb() 関数は、指定のデータベース名、コード・セット、モードを使用してデータベースを作成します。

仕様:

- **CLI** V9.7
- **ODBC**
- **ISO CLI**

SQLCreateDb API を実行するには、その前にサーバーへのアクティブな接続が存在している必要があります。

Unicode 環境での同等機能: これに対応する Unicode 関数は、SQLCreateDbW() 関数です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 31. SQLCreateDb 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*hDbc*	入力	接続ハンドル。
SQLCHAR *	*szDbName*	入力	作成するデータベースの名前。
SQLINTEGER	*cbDbName*	入力	*szDbName* 引数、または *szDbName* 引数がヌル終了ストリングの場合は SQL_NTS を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合には SQLWCHAR エレメント) の数。
SQLCHAR *	*szCodeSet*	入力	データベース・コード・セット情報。 **注:** *szCodeSet* 引数の値が NULL の場合、DB2 データ・サーバーのデータベースは Unicode コード・ページで作成され、IDS データ・サーバーのデータベースは UTF-8 コード・ページで作成されます。
SQLINTEGER	*cbCodeSet*	入力	*szCodeSet* 引数、または *szCodeSet* 引数がヌル終了ストリングの場合は SQL_NTS を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合には SQLWCHAR エレメント) の数。
SQLCHAR *	*szMode*	入力	データベース・ロギング・モード。 **注:** この値は、IDS データ・サーバーに対してのみ適用されます。
SQLINTEGER	*cbMode*	入力	*szMode* 引数、または *szMode* 引数がヌル終了ストリングの場合は SQL_NTS を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合には SQLWCHAR エレメント) の数。

使用法

DB2 データベースを作成する際、まず ATTACH キーワードを指定して、CLI アプリケーションをサーバー・インスタンスに接続させる必要があります。ATTACH キーワードを使用してサーバー・インスタンスに接続した後に使用できる API は、SQLCreateDb()、SQLDropDb()、および SQLDisconnect() です。新しいデータベース

で他の CLI 操作を実行する前に、サーバー・インスタンスから切断して新しいデータベースに接続する必要があります。

戻りコード

- SQL_SUCCESS
- SQL_ERROR

診断

表 32. SQLCreateDb SQLSTATE

SQLSTATE	説明	解説
08003	接続がクローズされています。	*SQLCreateDb* 引数で指定された接続がオープンしていませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	*cbDbName*、*cbCodeSet*、および *cbMode* 引数の最大長は 128 です。無効値を指定すると、CLI はエラーを生成します。

制限

- インスタンス接続を示す接続が必要です。
- SQLCreateDb() 関数は、DB2 for IBM i および DB2 for z/OS サーバーではサポートされません。

.

例

次の例では、DB2 データベースをローカル・サーバーに作成します。

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

次の例では、DB2 データベースをリモート・サーバーに作成します。

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

バージョン情報

最終更新

このトピックの最終更新対象は、IBM DB2 バージョン 9.7 フィックスパック 3 です。

IBM Data Server Client

IBM DB2 for Linux, UNIX, and Windows でサポート

SQLCreatePkg

SQLCreatePkg() はバインド・ユーティリティーを呼び出します。これは、バインド・ファイルに保管された SQL ステートメントを準備し、データベースに保管されるパッケージを作成します。

仕様:

- **CLI** 9.5

構文

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

関数引数

表 33. SQLCreatePkg() 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*hDbc*	入力	接続ハンドル。
SQLCHAR*	*szBindFileNameIn*	入力	バインドするファイルの名前もしくはバインド・ファイル名のリストを含むファイルの名前。
SQLINTEGER	*cbBindFileNameIn*	入力	*szBindFileNameIn* を保管するのに必要な SQLCHAR エレメント数、または *szBindFileNameIn* がヌルで終了する場合は SQL_NTS エレメント数。
SQLCHAR*	*szBindOpts*	入力	バインド・オプションのリスト (セミコロンで区切る)。
SQLINTEGER	*cbBindOpts*	入力	*szBindOpts* を保管するのに必要な SQLCHAR エレメント数、または *szBindOpts* がヌルで終了する場合は SQL_NTS エレメント数。

使用法

引数 *szBindFileNameIn* とは、バインド・ファイルの名前、もしくはバインド・ファイル名のリストを含むファイルの名前を含むストリングです。 バインド・ファイル名には拡張子 .bnd が付いている必要があります。 これらのファイルに対してパスを指定できます。 バインド・リスト・ファイルの名前の先頭にアットマーク (@) を付けます。 次の例は完全に修飾されたバインド・リストのファイル名です。

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

バインド・リスト・ファイルには 1 つ以上のバインド・ファイル名が含まれ、拡張子が .lst である必要があります。 最初のバインド・ファイル名以外のすべてのバインド・ファイル名の前に正符号 (+) を付けてください。 バインド・ファイル名が 1 つ以上の行にわたることも可能です。 例えば、バインド・リスト・ファイル all.lst には次に示す行を含めることができます。

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

リスト・ファイル内のバインド・ファイル名に対してパス指定を使用できます。 パスが指定されていない場合、データベース・マネージャーはバインド・リスト・ファイルからパス情報を取得します。

以下の **BIND** コマンド・パラメーターを SQLCreatePkg() とともに指定できます。

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

BIND コマンド・パラメーターは、セミコロンで区切った、名前と値の対のストリングの形で渡すことができます。例えば、以下のようにします。

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

オプションと値については両方とも、大/小文字は区別されません。

例 1: **REOPT**=ONCE および **ENCODING**=CCSID を指定したファイルのバインド

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

例 2: **KEEPDYNAMIC**=YES、**BLOCKING**=NO、および **ISOLATION**=RS を指定したすべてのリスト・ファイルのバインド

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

例 3: **COLLECTION**=SCHEMA NAME を指定したファイルのバインド

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

SQLDataSources 関数 (CLI) - データ・ソースのリストの取得

一度に 1 回ずつターゲット・データベースのリストを返します。

データベースを使用できるようにカタログする必要があります。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

SQLDataSources() は、通常、接続が行われる前に呼び出されて、接続に使用できるデータベースを判別します。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLDataSourcesW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

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

関数引数

表 34. SQLDataSources 引数

データ・タイプ	引数	使用法	説明
SQLHENV	*EnvironmentHandle*	入力	環境ハンドル。
SQLUSMALLINT	*Direction*	入力	アプリケーションはこれを使用して、リスト内の最初のデータ・ソース名を要求するか、またはリスト内の次のデータ・ソース名を要求します。 *Direction* は、以下の値のどちらかです。 • SQL_FETCH_FIRST • SQL_FETCH_NEXT
SQLCHAR *	*ServerName*	出力	データ・ソース名を戻す先のバッファーを指すポインター。
SQLSMALLINT	*BufferLength1*	入力	*ServerName* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。その値は、SQL_MAX_DSN_LENGTH + 1 以下でなければなりません。
SQLSMALLINT *	*NameLength1Ptr*	出力	**ServerName* に戻すために使用できる SQLCHAR エレメントの合計数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの合計数) を戻すバッファーを指すポインター (NULL 終止符文字を除く)。戻り値に使用できる SQLCHAR または SQLWCHAR エレメントの数が *BufferLength1* 以上の場合、**ServerName* 内のデータ・ソース名は、*BufferLength1* から NULL 終止符文字分を差し引いた長さに切り捨てられます。
SQLCHAR *	*Description*	出力	データ・ソースの記述が返されるバッファーを指すポインター。 CLI は、 DBMS にカタログ作成されたデータベースに関連した***注釈*** フィールドを返します。
SQLSMALLINT	*BufferLength2*	入力	*Description* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。

表 34. SQLDataSources 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*NameLength2Ptr*	出力	**Description* に戻すために使用できる SQLCHAR エレメントの合計数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの合計数) を戻すバッファーを指すポインター (NULL 終止符文字を除く)。戻り値に使用できる SQLCHAR または SQLWCHAR の数が *BufferLength2* 以上の場合、 **Description* 内のドライバー記述は、 *BufferLength2* から NULL 終止符文字分を差し引いた長さに切り捨てられます。

使用法

アプリケーションは、 SQL_FETCH_FIRST または SQL_FETCH_NEXT のいずれかに設定された *Direction* を使用して、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、常にリスト内の最初のデータベースが返されます。

SQL_FETCH_NEXT を指定すると、以下のようになります。
- SQL_FETCH_FIRST 呼び出しの直後に、リスト内の 2 番目のデータベースが返されます。
- 他のすべての SQLDataSources() 呼び出しの前に、リスト内の最初のデータベースが返されます。
- リスト内にデータベースがそれ以上ないと、SQL_NO_DATA_FOUND が返されます。関数を再度呼び出すと、最初のデータベースが返されます。
- その他の場合は、リスト内の次のデータベースが戻されます。

ODBC 環境では、ODBC Driver Manager がこの関数を実行します。

IBM RDBMS は常にデータ・ソースの記述 (30 バイトになるまでブランク埋め込み) を返すので、 CLI も同じようにします。

戻りコード
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

診断

表 35. SQLDataSources SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	引数 *ServerName* に返されたデータ・ソース名が、引数 *BufferLength1* に指定した値よりも長い値でした。引数 *NameLength1Ptr* には、完全データ・ソース名の長さが含まれています。 (関数は、SQL_SUCCESS_WITH_INFO を返します。) 引数 *Description* に返されたデータ・ソース名が、引数 *BufferLength2* に指定した値よりも長い値でした。引数 *NameLength2Ptr* には、完全データ・ソース記述の長さが含まれています。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY000	一般エラーです。	特定の SQLSTATE が用意されておらず、しかもインプリメンテーション独自の SQLSTATE も定義されていないエラーが発生しました。 SQLGetDiagRec() から *MessageText* 引数内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength1* に指定された値は、0 より小さい値でした。 引数 *BufferLength2* に指定された値は、0 より小さい値でした。
HY103	Direction オプションが範囲外です。	引数 *Direction* に指定された値は、 SQL_FETCH_FIRST または SQL_FETCH_NEXT に等しい値ではありませんでした。

許可

なし。

例

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

SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す

照会で生成された結果セット内で指定された列に共通に使用される記述子情報のセット (列名、タイプ、精度、スケール、NULL 可能) を返します。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

この情報は、IRD のフィールドでも使用できます。

アプリケーションが記述子情報の属性を 1 つだけ必要としているか、または SQLDescribeCol() によっては返されない属性を必要とする場合には、SQLDescribeCol() の代わりに SQLColAttribute() 関数を使用することができます。

この関数を呼び出す前に、 SQLPrepare() または SQLExecDirect() を呼び出す必要があります。

この関数 (または SQLColAttribute()) は通常、バインド列関数 (SQLBindCol() 、SQLBindFileToCol()) の前に呼び出され、アプリケーション変数にバインドする前に列の属性を判別します。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLDescribeColW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

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

関数引数

表 36. SQLDescribeCol 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*ColumnNumber*	入力	記述される列番号。列は、1 を最初の番号として順次左から右へ番号が付けられます。また、0 に設定してブックマーク列を記述することもできます。
SQLCHAR *	*ColumnName*	出力	列名バッファーを指すポインター。この値は、IRD の SQL_DESC_NAME フィールドから読み取られます。列名が不明なときは、これを NULL に設定してください。
SQLSMALLINT	*BufferLength*	入力	**ColumnName* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。

表 36. SQLDescribeCol 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*NameLengthPtr*	出力	**ColumnName* に戻すために使用できる SQLCHAR エレメントの合計数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの合計数) を戻すバッファーを指すポインター (NULL 終止符文字を除く)。*NameLengthPtr* が *BufferLength* 以上である場合、列名 (* *ColumnName*) は *BufferLength* - 1 (SQLCHAR または SQLWCHAR エレメントの個数) に切り捨てられます。
SQLSMALLINT *	*DataTypePtr*	出力	列の基本 SQL データ・タイプ。列に関連付けられているユーザー定義タイプがあるかどうかを判別するには、 *fDescType* を SQL_COLUMN_DISTINCT_TYPE に設定して、SQLColAttribute() を呼び出してください。サポートされるデータ・タイプについては、記号データ・タイプおよびデフォルト・データ・タイプの表の「記号 SQL データ・タイプ」列を参照してください。
SQLULEN *	*ColumnSizePtr*	出力	データベースで定義されている列の精度。 *fSqlType* が GRAPHIC または DBCLOB SQL データ・タイプを指示する場合、この変数は列が保持できる 2 バイト文字 の最大数を示します。
SQLSMALLINT *	*DecimalDigitsPtr*	出力	データベースで定義されている列のスケール (SQL_DECIMAL、SQL_NUMERIC、SQL_TYPE_TIMESTAMP だけに適用される)。個々の SQL データ・タイプのスケールについては、『データ・タイプ・スケール』の表を参照してください。
SQLSMALLINT *	*NullablePtr*	出力	この列に NULL が使用できるかどうかを示します。 • SQL_NO_NULLS • SQL_NULLABLE

使用法

列は、順次左から右へ割り当てられた番号で識別されます。記述はどの順序でも行うことができます。

- ブックマークを使用していない場合 (SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF)、列番号は 1 から始まります。
- ブックマークを使用している場合 (そのステートメント属性が SQL_UB_ON)、*ColumnNumber* 引数を 0 に設定してそのブックマーク列を記述することができます。

ポインター引数に NULL ポインターを指定すると、 CLI はアプリケーションが情報を要求していないとみなし、何も返しません。

列がユーザー定義タイプの場合、 SQLDescribeCol() は *DataTypePtr* に組み込みタイプだけを返します。ユーザー定義タイプを入手するには、*fDescType* を SQL_COLUMN_DISTINCT_TYPE に設定して、 SQLColAttribute() を呼び出してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

SQLDescribeCol() が SQL_ERROR または SQL_SUCCESS_WITH_INFO を返した場合、 SQLGetDiagRec() または SQLGetDiagField() 関数を呼び出して以下の SQLSTATE のうちの 1 つを取得することができます。

表 37. SQLDescribeCol SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	引数 * *ColumnName* 内に返された列名は、引数 *BufferLength* 内で指定された値より長い値でした。引数 * *NameLengthPtr* には、完全列名の長さが入っています。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
07005	ステートメントが結果セットを返しませんでした。	*StatementHandle* に関連したステートメントが結果セットを返しませんでした。記述する列がありませんでした。 (結果セット内に行があるかどうかを判別するには、まず SQLNumResultCols() を呼び出します。)
07009	無効な記述子索引	*ColumnNumber* に指定された値が 0 と同等であり、SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF でした。引数 *ColumnNumber* に指定された値は、結果セット内の列数より大きい値でした。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	SQLPrepare() または SQLExecDirect() を *StatementHandle* 用に呼び出す前に、この関数が呼び出されました。 実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。

表 37. SQLDescribeCol SQLSTATE (続き)

SQLSTATE	説明	解説
HY090	ストリングまたはバッファーの長さが無効です。	1 より小さい、引数 *BufferLength* で指定された長さ。
HYC00	ドライバーが使用できません。	列 *ColumnNumber* の SQL データ・タイプは、CLI では認識されません。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

以下の ODBC 定義のデータ・タイプはサポートされていません。

- SQL_BIT
- SQL_TINYINT

例

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

SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す

準備済み SQL ステートメントに関連したパラメーター・マーカーの記述を返します。

仕様:

- **CLI** 5.0
- **ODBC** 1.0
- **ISO CLI**

パラメーター・マーカーの記述は、IPD のフィールドでも使用できます。

据え置きの準備済みが使用可能になっているときに、初めて SQLDescribeParam()、SQLNumResultCols()、または SQLDescribeCol() を呼び出した場合、その呼び出しによって SQL ステートメントの PREPARE が強制的にサーバーに送られます。

構文

```
SQLRETURN   SQLDescribeParam (
               SQLHSTMT          StatementHandle,     /* hstmt */
               SQLUSMALLINT      ParameterNumber,     /* ipar */
               SQLSMALLINT       *DataTypePtr,        /* pfSqlType */
```

```
SQLULEN           *ParameterSizePtr,   /* pcbParamDef */
SQLSMALLINT       *DecimalDigitsPtr,   /* pibScale */
SQLSMALLINT       *NullablePtr);       /* pfNullable */
```

関数引数

表 38. SQLDescribeParam 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*ParameterNumber*	入力	パラメーター・マーカー番号は、パラメーターの小さい順に配列されていて、1 から始まっています。
SQLSMALLINT *	*DataTypePtr*	出力	パラメーターの SQL データ・タイプを返すバッファーを指すポインターです。この値は、IPD の SQL_DESC_CONCISE_TYPE レコード・フィールドから読み取られます。 ColumnNumber が 0 (ブックマーク列の場合) に等しいときは、 SQL_BINARY が可変長ブックマークの **DataTypePtr* に返されます。
SQLULEN *	*ParameterSizePtr*	出力	データ・ソースで定義されているとおりに、対応するパラメーター・マーカーの列または式のサイズを返す先のバッファーを指すポインターです。
SQLSMALLINT *	*DecimalDigitsPtr*	出力	データ・ソースで定義されているとおりに、対応するパラメーターの列または式の小数桁数を返す先のバッファーを指すポインターです。
SQLSMALLINT *	*NullablePtr*	出力	パラメーターが NULL を許可するかどうかを示す値を返すバッファーを指すパラメーターです。この値は、IPD の SQL_DESC_NULLABLE フィールドから読み取られます。 ODBC 仕様は、以下の戻り値をリストします。しかし、CLI ドライバーは SQL_NULLABLE_UNKNOWN 戻り値のみ戻します。 • SQL_NO_NULLS: NULL は許可しません (これがデフォルト値です)。 • SQL_NULLABLE: NULL を許可します。 • SQL_NULLABLE_UNKNOWN: NULL を許可するかどうかは判別できません。 **注:** CLI ドライバーは SQL_NULLABLE_UNKNOWN を返します。

使用法

パラメーター・マーカーには、SQL ステートメントに表示される順番で、小さい順に 1 からパラメーター番号が付けられます。

SQLDescribeParam() は、 SQL ステートメントのパラメーターのタイプ (入力、入出力、または出力) は返しません。ストアード・プロシージャーでの呼び出し以外では、 SQL ステートメントにあるパラメーターはすべて入力パラメーターです。ストアード・プロシージャーの呼び出しでの各パラメーターのタイプを判別するには、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

診断

表 39. SQLDescribeParam SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
07009	記述子索引が無効です。	引数 *ParameterNumber* に指定された値は、1 より小さい値でした。 引数 *ParameterNumber* に指定された値は、関連する SQL ステートメントのパラメーターより大きい値でした。 パラメーター・マーカーは、非 DML ステートメントの一部でした。 パラメーター・マーカーは、SELECT リストの一部でした。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
21S01	挿入値リストが列リストに一致していません。	INSERT ステートメントにあるパラメーターの数が、ステートメントで名前の指定された表にある列の数と一致しませんでした。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 `SQLCancel()` がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。

表 39. SQLDescribeParam SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	SQLPrepare() または SQLExecDirect() を *StatementHandle* 用に呼び出す前に、この関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 SQLExecute() SQLExecDirect()、SQLBulkOperations()、または SQLSetPos() が *StatementHandle* で呼び出され、SQL_NEED_DATA を戻しました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	基礎メモリー・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でした。メモリー不足状態が原因と考えられます。
HYC00	ドライバーが使用できません。	スキーマ関数ストアード・プロシージャーは、サーバー上ではアクセスできません。スキーマ関数ストアード・プロシージャーをサーバー上にインストールして、それらがアクセス可能であることを確認してください。

制限

なし。

SQLDisconnect 関数 (CLI) - データ・ソースからの切断

データベース接続ハンドルに関連した接続をクローズします。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

この接続に未解決のトランザクションがある場合、SQLDisconnect() を呼び出す前に、 SQLEndTran() を呼び出す必要があります。

この関数を呼び出した後で、SQLConnect() を呼び出して別のデータベースに接続するか、または SQLFreeHandle() を使って接続ハンドルを解放します。

構文

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

関数引数

表 40. SQLDisconnect 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル

使用法

アプリケーションが接続に関連したすべてのステートメント・ハンドルを解放する前に SQLDisconnect() を呼び出すと、 CLI はデータベースから正常に切断した後にそれらのステートメント・ハンドルを解放します。

SQL_SUCCESS_WITH_INFO が返された場合、それは、データベースからの切断は正常に完了したけれども、追加のエラーやインプリメンテーション独自の情報があることを意味します。例えば、切断以後のクリーンアップで問題が発生した場合や、アプリケーションとは無関係に発生したイベント (通信障害など) が原因で現行の接続がない場合があります。

SQLDisconnect() 呼び出しに成功した後、アプリケーションは *ConnectionHandle* を再利用して別の SQLConnect() または SQLDriverConnect() 要求を行うことができます。

アプリケーションは、SQLDisconnect() によってカーソルをクローズしてはなりません (ストアード・プロシージャーの場合も通常のクライアント・アプリケーションの場合も同様)。どちらの場合も、SQLCloseCursor() を使用してカーソルをクローズしてから、 SQLFreeHandle() を使用してステートメント・ハンドルを解放してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 41. SQLDisconnect SQLSTATE

SQLSTATE	説明	解説
01002	切断エラーです。	切断時にエラーが発生しました。しかし、切断は成功しました。(関数は、SQL_SUCCESS_WITH_INFO を返します。)
08003	接続がクローズされています。	引数 *ConnectionHandle* で指定された接続がオープンしていませんでした。
25000 **25**501	トランザクション状態が無効です。	引数 *ConnectionHandle* で指定された接続に処理中のトランザクションがありました。トランザクションはアクティブであり、接続を切断できません。 **注:** このエラーは、CLI で作成されたストアード・プロシージャーには適用されません。
25501	トランザクション状態が無効です。	引数 *ConnectionHandle* で指定された接続に処理中のトランザクションがありました。トランザクションはアクティブであり、接続を切断できません。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 41. SQLDisconnect SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。

制限

なし。

例

```
SQLHANDLE hdbc; /* connection handle */

/* ... */

/* disconnect from the database */
  cliRC = SQLDisconnect(hdbc);
```

SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続

SQLConnect() の代わりの関数。両方の関数ともターゲット・データベースに対する接続を確立しますが、 SQLDriverConnect() は追加接続パラメーターと、接続情報をユーザーに入力要求する機能をサポートします。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

SQLConnect() でサポートされる 3 つの入力引数 (データ・ソース名、ユーザー ID、およびパスワード) 以外のパラメーターがデータ・ソースに必要な場合、または CLI のグラフィカル・ユーザー・インターフェースを使用してユーザーに必須の接続情報を入力要求する場合に、SQLDriverConnect() を使用します。

接続が確立されると、完全な接続ストリングが返されます。アプリケーションは、以後の接続要求のためにこのストリングを保管することができます。

構文

汎用

```
SQLRETURN  SQLDriverConnect (
        SQLHDBC       ConnectionHandle,              /* hdbc */
        SQLHWND       WindowHandle,                  /* hwnd */
        SQLCHAR       *InConnectionString,           /* szConnStrIn */
        SQLSMALLINT   InConnectionStringLength,      /* cbConnStrIn */
        SQLCHAR       *OutConnectionString,          /* szConnStrOut */
        SQLSMALLINT   OutConnectionStringCapacity,   /* cbConnStrOutMax */
        SQLSMALLINT   *OutConnectionStringLengthPtr, /* pcbConnStrOut */
        SQLUSMALLINT  DriverCompletion);             /* fDriverCompletion */
```

関数引数

表 42. SQLDriverConnect 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル
SQLHWND	*WindowHandle*	入力	ウィンドウ・ハンドル。 Windows オペレーティング・システムでは、これは親 Windows ハンドルです。現在ウィンドウ・ハンドルは、Windows でのみサポートされています。 NULL が渡されると、ダイアログは表示されません。
SQLCHAR *	*InConnectionString*	入力	完全、一部、または空 (NULL ポインター) の接続ストリング (以下の構文と説明を参照)。
SQLSMALLINT	*InConnectionStringLength*	入力	*InConnectionString* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLSMALLINT *	*OutConnectionString*	出力	完全な接続ストリングのバッファーを指すポインター。 接続が正常に確立されると、このバッファーには完全な接続ストリングが入れられます。アプリケーションは、このバッファー用に少なくとも SQL_MAX_OPTION_STRING_LENGTH バイトを割り振る必要があります。
SQLSMALLINT	*OutConnectionString Capacity*	入力	*OutConnectionString* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLSMALLINT *	*OutConnectionString LengthPtr*	出力	*OutConnectionString* バッファーに戻すために使用できる SQLCHAR エレメントの数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの数) へのポインター (NULL 終止符文字を除く)。 **OutConnectionStringLengthPtr* の値が *OutConnectionStringCapacity* 以上である場合、*OutConnectionString* 内の完全接続ストリングは SQLCHAR または SQLWCHAR の個数が *OutConnectionStringCapacity* -1 になるように切り捨てられます。
SQLUSMALLINT	*DriverCompletion*	入力	CLI がいつ詳細情報をユーザーに要求すべきかを示します。 有効値は以下のとおりです。 • SQL_DRIVER_PROMPT • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE_REQUIRED • SQL_DRIVER_NOPROMPT

使用法

InConnectionString 引数

要求の接続ストリングは、以下の構文になります。

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

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

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

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

説明
- character-string には 0 個以上の SQLCHAR または SQLWCHAR エレメントが入れられます。
- identifier には 1 個以上の SQLCHAR または SQLWCHAR エレメントが入れられます。
- attribute-keyword は大文字小文字を区別しません。
- attribute-value は大文字小文字を区別することがあります。
- **DSN** キーワードの値はブランクのみでは成立しません。
- **NEWPWD** は、パスワード変更要求の一部として使用されます。アプリケーションは、`NEWPWD=anewpass;` などとして使用する新しいストリングを指定するか、または `NEWPWD=;` を指定して CLI ドライバーによって生成されるダイアログ・ボックスが新しいパスワードの入力を要求するようにすることができます。

接続ストリングと初期設定ファイルの文法上の理由から、**[]{}(),;?*=!@** 文字の入っているキーワードおよび属性値は避ける必要があります。システム情報の文法上、キーワードとデータ・ソース名には、円記号 (¥) を入れることができません。 CLI バージョン 2 の場合、**DRIVER** キーワードの前後に中括弧が必要です。

あるキーワードがブラウズ要求の接続ストリングの中で繰り返される場合、 CLI は、最初に現れたものの値を使用します。 **DSN** および **DRIVER** キーワードが同じブラウズ要求の接続ストリング内にある場合、 CLI は、最初に現れたキーワードの方を使用します。

OutConnectionString 引数

結果の接続ストリングは、接続属性のリストになっています。接続属性は、属性キーワードとそれに対応する属性値から成っています。ブラウズ結果の接続ストリングは、以下の構文になります。

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

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

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

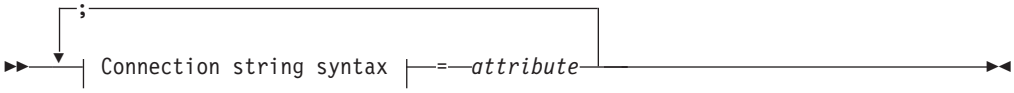
driver-defined-attribute-keyword ::= identifier[:localized-identifier]

attribute-value ::= {attribute-value-list} | ?
(中括弧はリテラルであり、CLI によって返されます。)
attribute-value-list ::= character-string [:localized-character
string] | character-string [:localized-character string], attribute-value-list

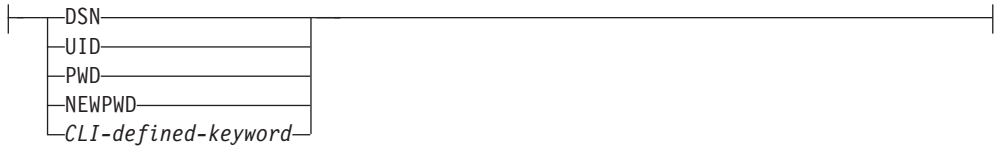
説明
- character-string と localized-character string には、 0 個以上の SQLCHAR または SQLWCHAR エレメントが入れられます。
- identifier および localized-identifier の SQLCHAR または SQLWCHAR エレメントの数は 1 以上です。 attribute-keyword は大文字小文字を区別しません。
- attribute-value は大文字小文字を区別することがあります。

接続ストリングと初期化ファイルの文法上の理由から、**[]{}(),;?*=!@** 文字の入っているキーワード、ローカライズ ID、および属性値は避ける必要があります。システム情報の文法上、キーワードとデータ・ソース名には、円記号 (¥) を入れることができません。

接続ストリングは、接続を完了するのに必要な 1 つ以上の値を渡すのに使用します。接続ストリングの内容と *DriverCompletion* の値で、 CLI がユーザーとダイアログを確立する必要があるかどうかを判別します。



Connection string syntax



各キーワードに関連付けられている属性は、次のとおりです。

DSN データ・ソース名。データベースの名前または別名。 *DriverCompletion* が SQL_DRIVER_NOPROMPT と等しいときに必要です。

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 のパスワードがないと、空の値が指定されます (PWD=;)。

NEWPWD
パスワード変更要求の一部として使用する新規パスワード。アプリケーションは、NEWPWD=anewpass; などで、使用する新しいストリングを指定するか、または NEWPWD=; を指定して CLI ドライバーによって生成されるダイアログ・ボックスが新しいパスワードの入力を要求するようにすることができます。 (*DriverCompletion* 引数には SQL_DRIVER_NOPROMPT 以外の値を指定。)

CLI キーワードの任意の 1 つを接続ストリング上に指定することができます。キーワードが接続ストリング内で繰り返し指定されると、キーワードの最初のオカレンスに関連した値が使用されます。

CLI 初期設定ファイルにキーワードがある場合、キーワードとそれらの値は、接続ストリングで CLI に渡される情報を追加するために使用されます。CLI 初期設定ファイル内の情報が接続ストリング内の情報と矛盾するときは、接続ストリング内の値が優先されます。

表示されたダイアログ・ボックスをエンド・ユーザーが*取り消す* と、SQL_NO_DATA_FOUND が返されます。

次に示す *DriverCompletion* の値で、ダイアログがいつオープンするかが決まります。

SQL_DRIVER_PROMPT:
ダイアログは常に開始されます。接続ストリングと CLI 初期設定ファイルからの情報は初期値として使用され、ダイアログ・ボックスで入力したデータによって補足されます。

SQL_DRIVER_COMPLETE:
ダイアログは、接続ストリング内の情報が不足しているときだけ開始されます。接続ストリングからの情報は初期値として使用され、ダイアログ・ボックスで入力したデータによって補足されます。

SQL_DRIVER_COMPLETE_REQUIRED:
ダイアログは、接続ストリング内の情報が不足しているときだけ開始されます。接続ストリングからの情報は、初期値として使用されます。必須情報しか要求されません。ユーザーは、必要な情報だけを要求されます。

SQL_DRIVER_NOPROMPT:
ユーザーは、情報を要求されません。接続ストリングに含まれている情報を使用して、接続が試行されます。情報が足りない場合、SQL_ERROR が返されます。

接続が確立されると、完全な接続ストリングが返されます。特定のユーザー ID で 1 つのデータベースに複数の接続をセットアップする必要のあるアプリケーションでは、この出力接続ストリングを保管する必要があります。次いで、このストリングを将来の SQLDriverConnect() 呼び出しの際の入力接続ストリング値として使用することができます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLDriverConnectW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA_FOUND
- SQL_INVALID_HANDLE
- SQL_ERROR

診断

SQLConnect() で生成されるすべての診断を、ここでも返すことができます。以下の表は、返すことのできる追加の診断を示したものです。

表 43. SQLDriverConnect SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	バッファー *szConnstrOut* は、接続ストリング全体を保留できるほど大きくありませんでした。引数 **OutConnectionStringLengthPtr* には、戻り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。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S00	接続ストリング属性が無効です。	入力接続ストリングに無効なキーワードまたは属性値が指定されましたが、以下にリストしたイベントのいずれかが発生したため、データ・ソースへの接続は成功しました。 • 認識されないキーワードが無視されました。 • 無効な属性値が無視され、その代わりにデフォルト値が使用されました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
HY000	一般エラーです。 ダイアログの失敗	接続ストリングに指定された情報は接続要求を行うには不十分でしたが、 *fCompletion* を SQL_DRIVER_NOPROMPT に設定してダイアログを禁止していました。 ダイアログを表示する試行が失敗しました。
HY090	ストリングまたはバッファーの長さが無効です。	*InConnectionStringLength* に指定された値は 0 より小さい値でしたが、 SQL_NTS と等しくありませんでした。 *OutConnectionStringCapacity* に指定された値は、0 より小さい値でした。
HY110	ドライバーの完了が無効です。	引数 *fCompletion* に指定された値は、有効値のいずれとも等しくありませんでした。

制限

なし。

例

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

SQLDropDb 関数 (CLI) - データベースのドロップ

SQLDropDb() 関数は、指定のデータベースをドロップします。

仕様:

- **CLI** V9.7

Unicode 環境での同等機能: これに対応する Unicode 関数は、SQLDropDbW() 関数です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 44. SQLDropDb 関数引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*hDbc*	入力	接続ハンドル。
SQLCHAR *	*szDbName*	入力	ドロップするデータベースの名前。
SQLINTEGER	*cbDbName*	入力	*szDbName* 引数、または *szDbName* 引数がヌル終了ストリングの場合は SQL_NTS を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合には SQLWCHAR エレメント) の数。

使用法

DB2 データベースをドロップするには、CLI アプリケーションはまず、ATTACH キーワードを使用してサーバー・インスタンスにアタッチする必要があります。ATTACH キーワードを使用してサーバー・インスタンスに接続した後に使用できる API は、 SQLCreateDb()、 SQLDropDb()、および SQLDisconnect() です。

戻りコード

- SQL_SUCCESS
- SQL_ERROR

制限

- 既に接続されているデータベースはドロップできません。
- SQLDropDb() 関数は、DB2 for IBM i および DB2 for z/OS データ・サーバーではサポートされません。

例

次の例では、DB2 データベースをローカル・サーバーで作成およびドロップします。

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

次の例では、DB2 データベースをリモート・サーバーで作成およびドロップします。

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

バージョン情報

最終更新

このトピックの最終更新対象は、IBM DB2 バージョン 9.7 フィックスパック 3 です。

IBM Data Server Client

IBM DB2 Database for Linux, UNIX, and Windows でサポート

SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了

接続に関連したすべてのステートメントでのすべてのアクティブな操作に関してか、または環境に関連したすべての接続に関して、コミットまたはロールバック操作を要求します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

SQLEndTran() は、接続に関連したすべてのステートメントでのすべてのアクティブな操作に関してか、または環境に関連したすべての接続に関して、コミットまたはロールバック操作を要求します。

構文

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

関数引数

表 45. SQLEndTran 引数

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*HandleType*	入力	*Handle* タイプ ID。 *Handle* が環境ハンドルの場合は SQL_HANDLE_ENV、または *Handle* が接続ハンドルの場合は SQL_HANDLE_DBC のどちらかが入ります。
SQLHANDLE	*ハンドル*	入力	タイプが *HandleType* によって示されるハンドルは、トランザクションの有効範囲を示します。
SQLSMALLINT	*CompletionType*	入力	以下の 2 つの値のどちらかです。 • SQL_COMMIT • SQL_ROLLBACK

使用法

HandleType が SQL_HANDLE_ENV であり、*Handle* が有効な環境ハンドルである場合、CLI はこの環境において接続状態にある接続に対して、一度に 1 つずつトランザクションをコミットするか、またはロールバックします。どちらを行うかは、*CompletionType* の値によって異なります。SQL_SUCCESS が返されるのは、各接続に SQL_SUCCESS を受け取るときだけです。1 つ以上の接続に対して SQL_ERROR を受け取る場合、アプリケーションに SQL_ERROR を返し、診断情報がこの環境の診断データ構造に入れられます。コミットまたはロールバック操作中に障害が生じた接続を判別するには、アプリケーションは各接続に対して SQLGetDiagRec() を呼び出すことができます。

分散作業単位環境を使用しているときは SQLEndTran() は使用できません。代わりにトランザクション・マネージャー API を使用してください。

CompletionType が SQL_COMMIT である場合、SQLEndTran() は、影響がある接続に関連しているあらゆるステートメントのアクティブな操作すべてに対してコミット要求を出します。*CompletionType* が SQL_ROLLBACK である場合、SQLEndTran() は、影響がある接続に関連しているあらゆるステートメントのアクティブな操作すべてに対してロールバック要求を出します。アクティブであるトランザクションがない場合は、SQLEndTran() は、データ・ソースに影響しないように SQL_SUCCESS を返します。

トランザクションがカーソルにどのように影響するかを判別するには、アプリケーションは SQLGetInfo() に SQL_CURSOR_ROLLBACK_BEHAVIOR と SQL_CURSOR_COMMIT_BEHAVIOR オプションを付けて呼び出します。

SQL_CURSOR_ROLLBACK_BEHAVIOR または SQL_CURSOR_COMMIT_BEHAVIOR 値が SQL_CB_DELETE と等しい場合、SQLEndTran() は、接続に関連しているすべてのステートメントにおいて、オープンしているカーソルすべてをクローズし、削除して、ペンディング中の結果をすべて破棄します。SQLEndTran() は、割り当てられている (準備されていない) 状態にあるステートメントは残します。アプリケーションはこれらを次の SQL 要求で再使用するか、あるいは SQLFreeStmt() または SQLFreeHandle() に SQL_HANDLE_STMT の *HandleType* を指定して呼び出し、割り振りを解除することができます。

SQL_CURSOR_ROLLBACK_BEHAVIOR または SQL_CURSOR_COMMIT_BEHAVIOR 値が SQL_CB_CLOSE と等しい場合、SQLEndTran() は、接続に関連しているすべてのステートメントにおいて、オープンしているカーソルすべてをクローズします。SQLEndTran() は、現存するすべてのステートメントを準備済み状態のままにします。これでアプリケーションは、接続に関連のあるステートメントの SQLExecute() を呼び出すときに、最初に SQLPrepare() を呼び出す必要がなくなります。

SQL_CURSOR_ROLLBACK_BEHAVIOR または SQL_CURSOR_COMMIT_BEHAVIOR 値が SQL_CB_PRESERVE と等しい場合、SQLEndTran() は、接続に関連している、オープンしているカーソルに影響しません。カーソルは、SQLEndTran() を呼び出す前に指していた行に残ります。

自動コミット・モードがオフになっているときに、トランザクションが 1 つもアクティブでない状態で SQL_COMMIT または SQL_ROLLBACK を指定して SQLEndTran() を呼び出した場合、SQL_SUCCESS が返されて、コミットまたはロールバックの対象となる作業がないことを示します。トランザクションに関連していないエラーが起きない限り、SQLEndTran() の呼び出しはデータ・ソースには何も影響を与えません。

自動コミット・モードがオンになっているときに、SQL_COMMIT か SQL_ROLLBACK のどちらかの *CompletionType* を指定して SQLEndTran() を呼び出した場合、トランザクションに関連していないエラーが起きない限り、常に SQL_SUCCESS が返されます。

CLI アプリケーションが自動コミット・モードで実行されている場合、 CLI ドライバーはステートメントをサーバーに渡しません。

ODBC ドライバーのバージョン 3.8 以降を使用するアプリケーションでは、SQLEnTran 関数は接続を延期状態に設定でき、(SQLSTATE を HY117 に設定して) SQL_ERROR を返します。必ず SQL_ATTR_ODBC_VERSION 環境属性を SQL_OV_ODBC3_80 に設定してください。接続を延期状態に設定するために必要な条件について詳しくは、Microsoft の MSDN 資料の SQLEndTran() を参照してください (http://msdn.microsoft.com/en-us/library/ms716544(v=vs.85).aspx)。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 46. SQLEndTran SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を返します。)
08003	接続がクローズされています。	*ConnectionHandle* は、接続状態にありません。
08007	トランザクション中に、接続に障害が起きました。	*ConnectionHandle* に関連した接続は関数の実行中に失敗しました。失敗の前に、要求された **COMMIT** または **ROLLBACK** が行われたかどうかは判別できません。
40001	トランザクションのロールバック	トランザクションは、別のトランザクションとのリソース・デッドロックのためにロールバックされます。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 46. SQLEndTran SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	非同期的に実行する関数が *ConnectionHandle* に関連している *StatementHandle* で呼び出され、SQLEndTran() を呼び出したときにもまだ実行中でした。 *ConnectionHandle* と関連する *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。 DB2 for z/OS データベース・サーバーに対して実行している CLI アプリケーションの場合は、この動作についての例外があります。接続属性 SQL_ATTR_FORCE_ROLLBACK がオンになっている場合、CLI アプリケーションは *CompletionType* が SQL_ROLLBACK になっているときに SQLEndTran() または SQLTransact() を正常に実行できます。StreamPutData 構成キーワードは 1 (オン) に設定する必要があります。
HY012	トランザクション・コードが無効です。	引数 *CompletionType* に指定された値は、SQL_COMMIT または SQL_ROLLBACK のどちらでもありません。
HY092	オプション・タイプが範囲外です。	引数 *HandleType* に指定された値は、 SQL_HANDLE_ENV または SQL_HANDLE_DBC のどちらでもありませんでした。

制限

なし。

例

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

/* ... */

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

/* ... */

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

SQLError 関数 (CLI) - エラー情報の取り出し

ODBC 3.0 では SQLError() は使用すべきでない関数なので、代わりに SQLGetDiagRec() と SQLGetDiagField() を使用します。

このバージョンの CLI でも引き続き SQLError() をサポートしていますが、最新の標準に準拠するように、SQLGetDiagRec() を CLI プログラムで使用します。

注:

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLErrorW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

新しい関数へのマイグレーション

ステートメント・ハンドルに関するエラー診断レコードを読むための SQLError() 関数は次のようなものでした。

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

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

SQLExecDirect 関数 (CLI) - ステートメントの直接実行

SQL ステートメント内にパラメーターが存在する場合、そのパラメーター・マーカー変数の現行値を使って、指定された SQL ステートメントまたは XQuery 式を直接実行します。

ステートメントまたは式は、1 回だけ実行することができます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

XQuery 式の場合、式そのものにパラメーター・マーカーを指定することはできません。しかし、XMLQUERY 関数を使用して、パラメーター・マーカーを XQuery 変数にバインドすることができます。次に、バインド済みパラメーター・マーカーの値は、実行のために、XMLQUERY で指定された XQuery 式に渡されます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLExecDirectW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 47. SQLExecDirect 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル *StatementHandle* に関連したオープン・カーソルがあってはなりません。

表 47. SQLExecDirect 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLCHAR *	*StatementText*	入力	SQL ステートメント・ストリングまたは XQuery 式のストリング。
SQLINTEGER	*TextLength*	入力	*StatementText* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *StatementText* がヌル終了ストリングの場合は SQL_NTS。

使用法

SQL ステートメント・テキストにベンダー・エスケープ節シーケンス列が入っている場合、 CLI はまず SQL ステートメント・テキストを適切な DB2 固有のフォーマットに修正してから、その作成および実行のためにそれをサブミットします。アプリケーションがベンダー・エスケープ節シーケンス列の入った SQL ステートメントを生成しない場合は、接続レベルで SQL_ATTR_NOSCAN ステートメント属性を SQL_NOSCAN_ON に設定することにより、 CLI がベンダー・エスケープ節を見つけるためにスキャンを実行しないようにします。

SQLExecDirect() を使って呼び出す場合の SQL ステートメントは、 COMMIT または ROLLBACK のどちらも使用できます。このようにすれば、現行接続ハンドルで SQLEndTran() を呼び出すのと同じ結果を生じます。

SQL ステートメント・ストリングには、パラメーター・マーカーが入っていることがありますが、 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おく必要があります。

SQL ステートメントが照会の場合、または *StatementText* が XQuery 式の場合、SQLExecDirect() はカーソル名を生成し、そのカーソルをオープンします。アプリケーションが SQLSetCursorName() を使用してカーソル名をステートメント・ハンドルに関連付けた場合、 CLI はアプリケーションで生成されたカーソル名を内部作成されたカーソル名に関連付けます。

結果セットが生成されると、SQLFetch() または SQLFetchScroll() は、バインドされた変数、LOB ロケーター、または LOB ファイル参照のいずれかの中に、その次の 1 つ以上のデータ行をフェッチします。

SQL ステートメントが定位置 DELETE または定位置 UPDATE の場合は、ステートメントで参照されるカーソルを行に入れる必要があり、また、同じ接続ハンドルのもとの別個のステートメント・ハンドルでそのカーソルを定義する必要があります。

ステートメント・ハンドル上にすでにオープンされているカーソルがあってはなりません。

SQL_ATTR_PARAMSET_SIZE 属性を指定して SQLSetStmtAttr() を呼び出して、入力パラメーター値の配列が各パラメーター・マーカーにバインドされていることを指定した場合、アプリケーションは、入力パラメーター値の配列全体を処理するのに SQLExecDirect() を一度しか呼び出す必要はありません。

ステートメントの実行によって複数の結果セット (入力パラメーターの各集まりごとに 1 つずつ) が返された場合は、現在の結果セットの処理が完了したときに、SQLMoreResults() を使用して次の結果セットに進まなければなりません。

戻りコード

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

アプリケーションが、 SQLBindParameter() のときに指定した *StrLen_or_IndPtr 値を 1 つ以上のパラメーター用に SQL_DATA_AT_EXEC に設定することで、実行時データ・パラメーター値を入力するよう要求すると、SQL_NEED_DATA が返されます。

SQL ステートメントが検索 UPDATE または検索 DELETE であり、検索条件を満たしている行がない場合、SQL_NO_DATA_FOUND が返されます。

診断

表 48. SQLExecDirect SQLSTATE

SQLSTATE	説明	解説
01504	UPDATE または DELETE ステートメントに、WHERE 節がありません。	*StatementText* の UPDATE ステートメントまたは DELETE ステートメントに、 WHERE 節が入っていませんでした。(表に行がなかった場合、関数は SQL_SUCCESS_WITH_INFO または SQL_NO_DATA_FOUND を返します。)
01508	ブロッキングには不適格なステートメントです。	このステートメントは、ストレージ以外の理由でブロッキングできませんでした。
07001	パラメーターの数が正しくありません。	SQLBindParameter() を使用してアプリケーション変数にバインドされたパラメーターの数が、引数 *StatementText* に含まれている SQL ステートメント内のパラメーター・マーカーの数より小さい値でした。
07006	無効な変換です。	CLI とアプリケーション変数の間でデータを転送すると、非互換のデータ変換が行われます。
21S01	挿入値リストが列リストに一致していません。	*StatementText* に INSERT ステートメントがあり、挿入する値の数が派生表の数と一致していませんでした。
21S02	派生表の次数が列リストに一致していません。	*StatementText* に CREATE VIEW ステートメントがあり、指定された名前の数は、照会指定で定義されている派生表と同じ数になっていません。
22001	ストリング・データの右側が切り捨てられました。	文字タイプ列に割り当てられた文字ストリングが、列の最大長を超えました。

表 48. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
22003	数値が範囲外です。	数値タイプ列に割り当てられた数値のために、割り当て時または中間結果の計算時に、数値の整数部分が切り捨てられました。 *StatementText* に、ゼロでの除算を行わせた算術式を含む SQL ステートメントがありました。 **注:** その結果、DB2 Database for Linux, UNIX, and Windows に対してカーソル状態は定義されません (他の RDBMS のカーソルはオープンしたままになります)。
22005	割り当てにエラーがありました。	*StatementText* にはパラメーターまたはリテラルのある SQL ステートメントが含まれており、値または LOB ロケーターは関連した表列のデータ・タイプとの互換性がありませんでした。 パラメーター値に関連付けられている長さ (SQLBindParameter()> に指定されている *pcbValue* バッファーの内容) は有効ではありません。 SQLBindParameter() または SQLSetParam() に使用されている引数 *fSQLType* は SQL GRAPHIC データ・タイプを表しますが、据え置き長さ引数 (*pcbValue*) には奇数の長さの値が入っています。GRAPHIC データ・タイプの場合、長さの値は偶数でなければなりません。
22007	無効な日付時刻形式です。	*StatementText* には無効な日時フォーマットの SQL ステートメントが入っていました。つまり、無効なストリング表示または値が指定されたか、あるいは値は無効な日付、時刻、またはタイム・スタンプのものです。
22008	日時フィールドがオーバーフローしました。	日時フィールドのオーバーフローが発生しました。例えば、日付またはタイム・スタンプの算術計算の結果が有効な日付範囲内の値にならないか、またはバインドされた変数が小さすぎるため日時値を代入できません。
22012	0 による除算は無効です。	*StatementText* に、ゼロでの除算を行わせた算術式を含む SQL ステートメントがありました。
23000	保全性制約違反です。	SQL ステートメントの実行ができないのは、それを実行すると DBMS 内に整合性制約違反が発生するからです。
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
24504	UPDATE、DELETE、SET、または GET ステートメントで識別されたカーソルが、行に位置付けられていません。	結果は直前の照会から *StatementHandle* でペンディングになったか、 *hstmt* に関連したカーソルがまだクローズされていませんでした。
34000	カーソル名が無効です。	*StatementText* に、位置指定された DELETE または位置指定された UPDATE がありますが、実行中のステートメントで参照されているカーソルはオープンされていませんでした。
37xxx [a]	SQL 構文が無効です。	*StatementText* に、以下のうちの 1 つ以上が入っていました。 • 接続されたデータベース・サーバーが準備できない SQL ステートメント • 構文エラーのあるステートメント

表 48. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
40001	トランザクションのロールバック	この SQL ステートメントが属するトランザクションは、デッドロックまたはタイムアウトが原因でロールバックされました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
42xxx	構文エラーまたはアクセス規則違反	**425**xx は、この許可 ID が、*StatementText* に含まれている SQL ステートメントを実行する権限を持っていないことを示します。 他の **42**xxx SQLSTATE は、構文の相違またはステートメントとのアクセス問題があることを示しています。

428A1	ホスト・ファイル変数によって参照されたファイルにアクセスできません。	これは、以下のどのシナリオでも起こる可能性があります。テキスト内で関連付けられた理由コードは、特定のエラーを表します。
		• 01 - ファイル名の長さが無効か、またはファイル名とパスの片方または両方のフォーマットが無効です。
		• 02 - ファイル・オプションが無効です。値は、以下のいずれかでなければなりません。
		`SQL_FILE_READ -既存ファイルからの読み取り`
		`SQL_FILE_CREATE -書き込みのための新規ファイルの作成`
		`SQL_FILE_OVERWRITE -既存ファイルの上書き ファイルが存在しない場合は ファイルを作成`
		`SQL_FILE_APPEND -既存ファイルへの付加 ファイルが存在しない場合は ファイルを作成`
		• 03 - ファイルが見つかりませんでした。
		• 04 - SQL_FILE_CREATE オプションが、既存のファイルと同じ名前を持つファイルに指定されました。
		• 05 - ファイルへのアクセスが拒否されました。ユーザーが、ファイルをオープンするための権限を持っていません。
		• 06 - ファイルへのアクセスが拒否されました。ファイルが非互換モードで使用中です。書き込まれるファイルが、排他モードでオープンされています。
		• 07 - ファイルへの書き込み中に、ディスクがフルになりました。
		• 08 - ファイルの読み取り中に、想定外のファイル終わりが見つかりました。
		• 09 - ファイルのアクセス中に、メディア・エラーが起きました。

42895	EXECUTE または OPEN ステートメント内のホスト変数の値は、そのデータ・タイプのために使用できません。	バインド・パラメーター関数呼び出しに指定された LOB ロケーター・タイプが、パラメーター・マーカーの LOB データ・タイプと一致していません。 バインド・パラメーター関数に使用される引数 *fSQLType* は、LOB ロケーター・タイプを指定しましたが、対応するパラメーター・マーカーは LOB ではありません。

表 48. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
44000	保全性制約違反です。	*StatementText* に、パラメーターまたはリテラルのある SQL ステートメントが含まれていました。このパラメーター値が、関連した表列で NOT NULL として定義されている列について NULL だったか、ユニーク値だけが入るように制約された列について重複値が指定されていたか、または、他の整合性制約に違反しました。
56084	DRDA® では、LOB データはサポートされていません。	ホストまたは IBM Power Systems™ サーバーに接続 (DB2 Connect™ を使用して行う) しているときは、 LOB 列の選択や更新を行うことができません。
58004	予期しないシステム障害です。	回復不能システム・エラー。
S0001	データベース・オブジェクトはすでにあります。	*StatementText* 内に、CREATE TABLE ステートメントまたは CREATE VIEW ステートメントがあり、指定されている表名またはビュー名はすでに存在しています。
S0002	データベース・オブジェクトはありません。	*StatementText* に、存在しない表名またはビュー名を参照する SQL ステートメントが含まれていました。
S0011	索引はすでにあります。	*StatementText* に CREATE INDEX ステートメントがあり、指定された索引名はすでに存在していました。
S0012	索引は見つかりません。	*StatementText* に DROP INDEX ステートメントがあり、指定された索引名は存在していませんでした。
S0021	列はすでにあります。	*StatementText* 内には ALTER TABLE ステートメントがありますが、 ADD 節に指定されている列はユニークな列ではなかったか、または基本表の既存の列を識別していました。
S0022	列は見つかりません。	*StatementText* に、存在しない列名を参照する SQL ステートメントが含まれていました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	*StatementText* は、NULL ポインターでした。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *TextLength* は 1 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HY092	オプション・タイプが範囲外です。	直前の `SQLBindFileToParam()` 操作の *FileOptions* 引数が無効でした。
HY503	ファイル名の長さが無効です。	`SQLBindFileToParam()` からの *fileNameLength* 引数値は 0 より小さい値でしたが、 SQL_NTS と等しい値ではありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 `SQLSetStmtAttr()` の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

表 48. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
注:		
a	xxx は、そのクラス・コードの任意の SQLSTATE を表します。例えば、 **37**xxx は **37** クラスの任意の SQLSTATE を表します。	

制限

なし。

例

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

SQLExecute 関数 (CLI) - ステートメントの実行

同一のステートメント・ハンドルで SQLPrepare() を使用して正常に作成されたステートメントを 1 回以上実行します。

このステートメントの実行では、SQLBindParameter() または SQLBindFileToParam() によってパラメーター・マーカーにバインドされた任意のアプリケーション変数の現行値が使用されます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

構文

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

関数引数

表 49. SQLExecute 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル *StatementHandle* に関連したオープン・カーソルがあってはなりません。

使用法

SQLPrepare() を使って事前に *StatementHandle* で準備された SQL ステートメント・ストリングには、パラメーター・マーカーが入っていることがあります。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おく必要があります。

注: XQuery 式の場合、式そのものにパラメーター・マーカーを指定することはできません。しかし、XMLQUERY 関数を使用して、パラメーター・マーカーを

XQuery 変数にバインドすることができます。次に、バインド済みパラメーター・マーカーの値は、実行のために、XMLQUERY で指定された XQuery 式に渡されます。

アプリケーションが SQLExecute() 呼び出しからの結果を処理した後で、新しい (または同じ) パラメーター値で再度ステートメントを実行することができます。

SQLExecute() を呼び出しても、 SQLExecDirect() で実行されたステートメントを再実行することはできません。 SQLPrepare() を使って準備したステートメントだけを実行することができ、しかも SQLExecute() を使ってのみ再実行できます。

作成された SQL ステートメントが照会または XQuery 式の場合、SQLExecute() はカーソル名を生成し、そのカーソルをオープンします。アプリケーションが SQLSetCursorName() を使用してカーソル名をステートメント・ハンドルに関連付けた場合、 CLI はアプリケーションで生成されたカーソル名を内部作成されたカーソル名に関連付けます。

特定のステートメント・ハンドルで照会を複数回実行するには、アプリケーションは SQL_CLOSE オプションを指定して SQLCloseCursor() または 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。 SQLExecute() を呼び出すときに、ステートメント・ハンドルのカーソルがオープンしていてはなりません。

結果セットが生成されると、SQLFetch() または SQLFetchScroll() は、バインドされた変数、LOB ロケーター、または LOB ファイル参照のいずれかの中に、その次の 1 つ以上のデータ行をフェッチします。

SQL ステートメントが定位置 DELETE または定位置 UPDATE の場合は、SQLExecute() を呼び出すときに、ステートメントで参照されるカーソルを行に入れる必要があり、また、同じ接続ハンドルの別個のステートメント・ハンドルでそのカーソルを定義する必要があります。

SQL_ATTR_PARAMSET_SIZE 属性を指定して SQLSetStmtAttr() を呼び出して、入力パラメーター値の配列が各パラメーター・マーカーにバインドされていることを指定した場合、アプリケーションは、入力パラメーター値の配列全体を処理するのに SQLExecute() を一度しか呼び出す必要はありません。ステートメントの実行によって複数の結果セット (入力パラメーターの各集まりごとに 1 つずつ) が返された場合は、現在の結果セットの処理が完了してから、 SQLMoreResults() を使用して次の結果セットに進まなければなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA
- SQL_NO_DATA_FOUND

アプリケーションが、 SQLBindParameter() のときに指定した *StrLen_or_IndPtr* 値を 1 つ以上のパラメーター用に SQL_DATA_AT_EXEC に設定することで、実行時データ・パラメーター値を入力するよう要求すると、SQL_NEED_DATA が返されます。

SQL ステートメントが検索 UPDATE または検索 DELETE であり、検索条件を満たしている行がない場合、SQL_NO_DATA_FOUND が返されます。

診断

SQLExecute() の SQLSTATE は、SQLExecDirect() のすべての SQLSTATE で構成されています。ただし、**HY**009 と **HY**090 は除かれ、以下の表に示されている SQLSTATE が追加されています。SQLPrepare() から戻される可能性のあるすべての SQLSTATE は、SQLExecute() の呼び出しでも、据え置き準備の振る舞いの結果として戻される可能性があります。

表 50. SQLExecute SQLSTATE

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	指定された *StatementHandle* は準備済みではありませんでした。最初に SQLPrepare() を呼び出さずに、 SQLExecute() を呼び出しました。

許可

なし。

例

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

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

/* ... */

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

/* ... */

/* bind parameter1 to the statement */
cliRC = SQLBindParameter(hstmt,
                         1,
                         SQL_PARAM_INPUT,
                         SQL_C_SHORT,
                         SQL_SMALLINT,
                         0,
                         0,
                         &parameter1,
                         0,
                         NULL);
/* ... */
parameter1 = 15;

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

SQLExtendedBind 関数 (CLI) - 列の配列のバインド

SQLBindCol() または SQLBindParameter() を繰り返し呼び出さずに、列またはパラメーターの配列をバインドします。

SQLExtendedBind 関数 (CLI) - 列の配列のバインド

仕様:

- **CLI** 6

構文

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

関数引数

表 51. SQLExtendedBind() 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLSMALLINT	*fBindCol*	入力	SQL_TRUE の場合、結果は SQLBindCol() に類似したものになります。そうでない場合は、SQLBindParameter() に類似したものになります。
SQLSMALLINT	*cRecords*	入力	バインドする列またはパラメーターの数。
SQLSMALLINT *	*pfCType*	入力	アプリケーション・データ・タイプの値の配列。
SQLPOINTER *	*rgbValue*	入力	アプリケーション・データ域を指すポインターの配列。
SQLINTEGER *	*cbValueMax*	入力	*rgbValue* の最大サイズの配列。
SQLUINTEGER *	*puiPrecisionCType*	入力	小数点精度値の配列。いずれの値も、対応するレコードのアプリケーション・データ・タイプが SQL_C_DECIMAL_IBM の場合のみ使用します。
SQLSMALLINT *	*psScaleCType*	入力	小数桁数の配列。いずれの値も、対応するレコードのアプリケーション・データ・タイプが SQL_C_DECIMAL_IBM の場合のみ使用します。
SQLINTEGER **	*pcbValue*	入力	長さの値を指すポインターの配列。
SQLINTEGER **	*piIndicator*	入力	標識の値を指すポインターの配列。SQL_ATTR_EXTENDED_INDICATORS 属性を使用して拡張標識フィーチャーが有効になっている場合、*piIndicator* 引数を使用すると、メソッドを介して、SQL_UNASSIGNED 定数と SQL_DEFAULT_PARAM 定数を渡すことができます。

表 51. SQLExtendedBind() 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*pfParamType*	入力	パラメーター・タイプの配列。使用されるのは、*fBindCol* が FALSE の場合だけです。 この配列の各行は、SQLBindParameter() の引数 *InputOutputType* と同じ目的を果たします。以下の値に設定できます。 • SQL_PARAM_INPUT • SQL_PARAM_INPUT_OUTPUT • SQL_PARAM_OUTPUT
SQLSMALLINT *	*pfSQLType*	入力	SQL データ・タイプの配列。使用されるのは、*fBindCol* が FALSE の場合だけです。 この配列の各行は、SQLBindParameter() の引数 *ParameterType* と同じ目的を果たします。
SQLUINTEGER *	*pcbColDef*	入力	SQL 精度値の配列。使用されるのは、*fBindCol* が FALSE の場合だけです。 この配列の各行は、SQLBindParameter() の引数 *ColumnSize* と同じ目的を果たします。
SQLSMALLINT *	*pibScale*	入力	SQL スケール値の配列。使用されるのは、*fBindCol* が FALSE の場合だけです。 この配列の各行は、SQLBindParameter() の引数 *DecimalDigits* と同じ目的を果たします。

使用法

引数 *fBindCol* は、この関数呼び出しを以下のどちらの関連付け (バインド) に使用するかを決定します。

- SQL ステートメントのパラメーター・マーカー (SQLBindParameter() と同様) - *fBindCol* = SQL_FALSE
- 結果セットの列 (SQLBindCol() と同様) - *fBindCol* = SQL_TRUE

この関数は、SQLBindCol() または SQLBindParameter() に対する複数の呼び出しを置き換えるために使用できますが、重要な違いに注意してください。 *fBindCol* パラメーターの設定方法によって、SQLExtendedBind() への入力は、以下の例外を除いて SQLBindCol() または SQLBindParameter() に類似しています。

- SQLExtendedBind() が SQLBindCol() モードに設定されている場合。
  - *targetValuePtr* は、戻り列にあるデータの最大長をバイトで示す、正の整数である必要があります。
- SQLExtendedBind() が SQLBindParameter() モードに設定されている場合。
  - *ColumnSize* は、該当する場合、ターゲット列の最大長をバイトで示す、正の整数である必要があります。
  - *DecimalDigits* は、該当する場合、ターゲット列のための正しいスケールに設定する必要があります。
  - SQL_C_DEFAULT を *ValueType* に使用することはできません。

- *ValueType* がロケーター・タイプの場合、対応する *ParameterType* はマッチングするロケーター・タイプでなければなりません。
- すべての *ValueType* から *ParameterType* へのマッピングは、 CLI が実行しなければならない変換を最小化するために、可能なかぎり一致する必要があります。

SQLExtendedBind() に渡すどの配列参照にも、少なくとも *cRecords* で指示されている数のエレメントが入っていなければなりません。呼び出し側のアプリケーションが十分な大きさの配列を渡さなかった場合、 CLI は配列の末尾を超えて読み取りを試みることがあり、その場合はデータが壊れたり重要なアプリケーションに障害が起きたりすることになります。

SQLExtendedBind() に渡される各配列は、据え置き引数と見なされます。つまり、配列内の値は実行時にリトリーブされ、調べられます。結果として、各配列が有効な状態にあり、CLI が配列内の値を使用して実行する場合に、有効なデータが含まれていることを確認します。正常な実行に続いて、ステートメントを再実行する必要がある場合、元の呼び出しから SQLExtendedBind() へ渡されたハンドルが、まだ有効な配列を参照している場合、2 度目に SQLExtendedBind() を呼び出す必要はありません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 52. SQLExtendedBind() SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	*pfCType* 引数の行によって識別されるデータ値から *pfParamType* 引数によって識別されるデータ・タイプへの変換は、意味のある変換ではありません。 (例えば、SQL_C_TYPE_DATE から SQL_DOUBLE への変換です。)
07009	無効な記述子索引	引数 *cRecords* に指定された値が、結果セット内の列の最大数を超えました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY003	プログラム・タイプが範囲外です。	*pfParamType* または *pfSQLType* の行は、有効なデータ・タイプまたは SQL_C_DEFAULT ではありませんでした。
HY004	SQL データ・タイプが範囲外です。	引数 *pfParamType* に指定された値が、有効な SQL データ・タイプではありません。

表 52. SQLExtendedBind() SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引数の値が無効です。	引数 *rgbValue* は NULL ポインターで、引数 *cbValueMax* も NULL ポインターですが、 *pfParamType* が SQL_PARAM_OUTPUT ではありません。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY021	不整合な記述子情報	整合性チェック時にチェックされた記述子情報は、整合性がとれていませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *cbValueMax* に指定された値は 1 より小さく、 *pfParamType* または *pfSQLType* の対応する行は、SQL_C_CHAR、SQL_C_BINARY、または SQL_C_DEFAULT のいずれかです。
HY093	無効なパラメーター数です。	引数 *pfCType* の行に指定された値が、1 より小さいか、サーバーでサポートされる最大数より大きい値でした。
HY094	位取りの値が無効です。	*pfParamType* に指定された値が SQL_DECIMAL または SQL_NUMERIC であり、 *DecimalDigits* に指定された値が 0 より小さいかまたは引数 *pcbColDef* (精度) の値より大きい値でした。 *pfParamType* に指定された値が SQL_C_TYPE_TIMESTAMP で、*pfParamType* に指定された値が SQL_CHAR または SQL_VARCHAR のどちらかであり、 *DecimalDigits* に指定された値が 0 より小さいかまたは 9 より大きい値でした。 *pfParamType* に指定された値が SQL_C_TIMESTAMP_EXT で、DecimalDigits に指定された値が 0 より小さいかまたは 12 より大きい値でした。
HY104	精度の値が無効です。	*pfParamType* に指定された値が SQL_DECIMAL または SQL_NUMERIC のどちらかで、 *pcbColDef* によって指定された値が 1 より小さい値でした。
HY105	パラメーター・タイプが無効です。	*pfParamType* が 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	ドライバーが使用できません。	CLI は、 *pfParamType* または *pfSQLType* の行に指定されているデータ・タイプを認識はしますが、サポートしません。 LOB ロケーター C データ・タイプが指定されましたが、接続されているサーバーは LOB データ・タイプをサポートしていません。

制限

なし

SQLExtendedFetch 関数 (CLI) - 拡張フェッチ (行の配列のフェッチ)

ODBC 3.0 では SQLExtendedFetch() は使用すべきでない関数なので、代わりに SQLFetchScroll() を使用します。

このバージョンの CLI でも引き続き SQLExtendedFetch() をサポートしていますが、最新の標準に準拠するように、SQLFetchScroll() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

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

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

注:

SQLExtendedFetch() の *rowCount* および *rowStatus* パラメーターに戻される情報は、 SQLFetchScroll() によって次のように処理されます。

- *rowCount* : SQLFetchScroll() は、 SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性が指し示すバッファー内にフェッチされた行数を戻します。
- *rowStatus* : SQLFetchScroll() は、 SQL_ATTR_ROW_STATUS_PTR ステートメント属性が指し示すバッファー内の各行の状況配列を戻します。

SQLExtendedPrepare 関数 (CLI) - ステートメントの準備とステートメント属性の設定

ステートメントの準備とステートメント属性グループの設定を 1 回の呼び出しで実行します。

仕様:

- **CLI** 6.0

SQLPrepare() を 1 回呼び出してから SQLSetStmtAttr() を複数回呼び出す代わりに、この関数を使用できます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLExtendedPrepareW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN SQLExtendedPrepare(
            SQLHSTMT      StatementHandle,  /* hstmt */
            SQLCHAR       *StatementText,   /* pszSqlStmt */
            SQLINTEGER    TextLength,       /* cbSqlStmt */
            SQLINTEGER    cPars,
```

```
SQLSMALLINT   sStmtType,
SQLINTEGER    cStmtAttrs,
SQLINTEGER    *piStmtAttr,
SQLINTEGER    *pvParams );
```

関数引数

表 53. SQLExtendedPrepare() 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*StatementText*	入力	SQL ステートメント・ストリング。
SQLINTEGER	*TextLength*	入力	*StatementText* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *StatementText* がヌル終了ストリングの場合は SQL_NTS。
SQLINTEGER	*cPars*	入力	ステートメントのパラメーター・マーカーの数。
SQLSMALLINT	*cStmtType*	入力	ステートメント・タイプ。有効な値については、cStmtType 値のリストを参照してください。
SQLINTEGER	*cStmtAttrs*	入力	この呼び出しで指定するステートメント属性の数。
SQLINTEGER *	*piStmtAttr*	入力	設定するステートメント属性の配列。
SQLINTEGER *	*pvParams*	入力	設定する、対応するステートメント属性値の配列。

使用法

この関数の最初の 3 つの引数は、SQLPrepare() の引数とまったく同じです。

SQLExtendedPrepare() 使用時の要件は、以下の 2 つです。

1. SQL ステートメントをスキャンして、ODBC/ベンダーのエスケープ節を探さない。これは、 SQL_ATTR_NOSCAN ステートメント属性が SQL_NOSCAN に設定されている場合のような動作になります。 SQL ステートメントに ODBC/ベンダーのエスケープ節が含まれている場合、 SQLExtendedPrepare() は使用できません。
2. SQL ステートメントに挿入するパラメーター・マーカーの数を事前に (*cPars* で) 指示しておく。

cPars 引数は、*StatementText* 内のパラメーター・マーカーの数を指示します。

引数 *cStmtType* は、準備中のステートメントのタイプを指示する場合に使用します。有効な値のリストについては、cStmtType 値のリストを参照してください。

最後の 3 つの引数は、使用するステートメント属性のセットを指示する場合に使用します。この呼び出しで指定するステートメント属性には、*cStmtAttrs* を設定します。配列は、ステートメント属性リストの保持用とそれぞれの値の保持用に 2 つ作成してください。作成した配列を *piStmtAttr* と *pvParams* に使用します。

cStmtType 値のリスト

引数 *cStmtType* は、以下のいずれかの値に設定できます。

- SQL_CLI_STMT_UNDEFINED
- SQL_CLI_STMT_ALTER_TABLE

- SQL_CLI_STMT_CREATE_INDEX
- SQL_CLI_STMT_CREATE_TABLE
- SQL_CLI_STMT_CREATE_VIEW
- SQL_CLI_STMT_DELETE_SEARCHED
- SQL_CLI_STMT_DELETE_POSITIONED
- SQL_CLI_STMT_GRANT
- SQL_CLI_STMT_INSERT
- SQL_CLI_STMT_INSERT_VALUES
- SQL_CLI_STMT_REVOKE
- SQL_CLI_STMT_SELECT
- SQL_CLI_STMT_UPDATE_SEARCHED
- SQL_CLI_STMT_UPDATE_POSITIONED
- SQL_CLI_STMT_CALL
- SQL_CLI_STMT_SELECT_FOR_UPDATE
- SQL_CLI_STMT_WITH
- SQL_CLI_STMT_SELECT_FOR_FETCH
- SQL_CLI_STMT_VALUES
- SQL_CLI_STMT_CREATE_TRIGGER
- SQL_CLI_STMT_SELECT_OPTIMIZE_FOR_NROWS
- SQL_CLI_STMT_SELECT_INTO
- SQL_CLI_STMT_CREATE_PROCEDURE
- SQL_CLI_STMT_CREATE_FUNCTION
- SQL_CLI_STMT_SET_CURRENT_QUERY_OPT

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 54. SQLExtendedPrepare SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01504	UPDATE または DELETE ステートメントに、WHERE 節がありません。	*StatementText* の UPDATE ステートメントまたは DELETE ステートメントに、 WHERE 節が入っていませんでした。
01508	ブロッキングには不適格なステートメントです。	このステートメントは、ストレージ以外の理由でブロッキングできませんでした。
01S02	オプション値が変更されました。	CLI は **pvParams* の指定値をサポートしていないか、または **pvParams* の指定値が SQL の制約および要件にかなっていないため、 CLI が同等の値を代用しました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。

表 54. SQLExtendedPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
21S01	挿入値リストが列リストに一致していません。	*StatementText* に INSERT ステートメントがあり、挿入する値の数が派生表の数と一致していませんでした。
21S02	派生表の次数が列リストに一致していません。	*StatementText* に CREATE VIEW ステートメントがあり、指定された名前の数は、照会指定で定義されている派生表と同じ数になっていません。
22018	キャスト指定の文字値が無効です。	**StatementText* にリテラルまたはパラメーターを含む SQL ステートメントがあり、この値には関連した表の列のデータ・タイプとの互換がありませんでした。
22019	無効なエスケープ文字	引数 *StatementText* の WHERE 節に ESCAPE 付きの LIKE 述部があり、 ESCAPE の後に続くエスケープ文字の長さが 1 と等しくありませんでした。
22025	無効なエスケープ・シーケンス	引数 *StatementText* の WHERE 文節に「LIKE *パターン値* ESCAPE *エスケープ文字*」があり、パターン値のエスケープ文字の後の文字は「%」でも「_」でもありませんでした。
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
34000	カーソル名が無効です。	*StatementText* に、位置指定された DELETE または位置指定された UPDATE がありますが、実行中のステートメントで参照されているカーソルはオープンされていませんでした。
37xxx [a]	SQL 構文が無効です。	*StatementText* に、以下のうち 1 つ以上の問題が含まれていました。 • 接続されたデータベース・サーバーが準備できない SQL ステートメント • 構文エラーのあるステートメント
40001	トランザクションのロールバック	この SQL ステートメントが属するトランザクションは、デッドロックまたはタイムアウトが原因でロールバックされました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
42xxx [a]	構文エラーまたはアクセス規則違反	**42**5xx は、この許可 ID が、*StatementText* に含まれている SQL ステートメントを実行する権限を持っていないことを示します。 他の **42**xxx SQLSTATE は、構文の相違またはステートメントとのアクセス問題があることを示しています。
58004	予期しないシステム障害です。	回復不能システム・エラー。
S0001	データベース・オブジェクトはすでにあります。	*StatementText* 内に、CREATE TABLE ステートメントまたは CREATE VIEW ステートメントがあり、指定されている表名またはビュー名はすでに存在しています。
S0002	データベース・オブジェクトはありません。	*StatementText* に、存在していない表名またはビュー名を参照する SQL ステートメントがあります。
S0011	索引はすでにあります。	*StatementText* に CREATE INDEX ステートメントがあり、指定された索引名はすでに存在していました。
S0012	索引は見つかりません。	*StatementText* に DROP INDEX ステートメントがあり、指定された索引名は存在していませんでした。

表 54. SQLExtendedPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
S0021	列はすでにあります。	*StatementText* 内には ALTER TABLE ステートメントがありますが、 ADD 節に指定されている列はユニークな列ではなかったか、または基本表の既存の列を識別していました。
S0022	列は見つかりません。	*StatementText* に、存在していない列名を参照する SQL ステートメントがあります。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 `SQLCancel()` がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	*StatementText* は、NULL ポインターでした。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY011	この時点で無効な操作です。	*Attribute* は、SQL_ATTR_CONCURRENCY、SQL_ATTR_CURSOR_TYPE、SQL_ATTR_SIMULATE_CURSOR、または SQL_ATTR_USE_BOOKMARKS であり、ステートメントは準備済みでした。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY017	自動割り振りの記述子ハンドルについて無効な使用です。	*Attribute* 引数が SQL_ATTR_IMP_ROW_DESC または SQL_ATTR_IMP_PARAM_DESC でした。 *Attribute* 引数は SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC であり、 **ValuePtr* の値は、暗黙的に割り当てられた記述子ハンドルでした。
HY024	属性の値が無効です。	指定されている *Attribute* 値に対して、**ValuePtr* に無効値を指定しました。 (CLI がこの SQLSTATE を戻すのは、SQL_ATTR_ACCESS_MODE などの離散的な値セットを受け入れる接続およびステートメント属性に対してのみです。他のすべての接続およびステートメント属性については、ドライバーは **ValuePtr* で指定された値を検査する必要があります。)
HY090	ストリングまたはバッファーの長さが無効です。	引数 *TextLength* は 1 より小さい値でしたが、SQL_NTS と等しくありませんでした。

表 54. SQLExtendedPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
HY092	オプション・タイプが範囲外です。	引数 *Attribute* に指定された値が、このバージョンの CLI では無効なものでした。
HYC00	ドライバーが使用できません。	引数 *Attribute* に指定された値は、このバージョンの CLI ドライバーには有効な接続またはステートメント属性でしたが、データ・ソースではサポートされていませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

注:

a xxx は、そのクラス・コードの任意の SQLSTATE を表します。例えば、 **37**xxx は **37** クラスの任意の SQLSTATE を表します。

注: すべての DBMS が準備時に診断メッセージをすべて報告するわけではありません。据え置き準備がデフォルトの振る舞い (SQL_ATTR_DEFERRED_PREPARE ステートメント属性で制御します) としてオンのままになっていると、 PREPARE がサーバーに送られたときにこのようなエラーが発生する可能性があります。アプリケーションは、このような流れを生じる関数を呼び出すときにこれらの条件を処理できなければなりません。この種の関数には、SQLExecute()、SQLDescribeParam()、SQLNumResultCols()、SQLDescribeCol()、および SQLColAttribute() などがあります。

制限

IDS データ・サーバーにアクセスする場合は、 IDS データ・サーバー固有の SQLExtendedPrepare() 属性のみがサポートされます。 IDS データ・サーバーでサポートされない SQLExtendedPrepare() 属性を使用すると、「ドライバーが使用できません」というエラーが返されます。

SQLExtendedProcedures 関数 (CLI) - プロシージャー名のリストの取得

SQLExtendedProcedures() 関数は、サーバーに登録されていてしかも指定した検索パターンと一致するストアード・プロシージャー名のリストを戻します。

この情報は SQL 結果セットにして戻されます。これは、照会により生成された結果セットの処理で使用するものと同じ関数を用いて取り出すことができます。

仕様:

- **CLI** 9.7

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLExtendedProceduresW です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

モジュールとは、スキーマの概念を拡張したものです。 DB2 バージョン 9.7 以降のデータ・サーバーに接続するアプリケーションは、スキーマ内にモジュールを作

成し、そのモジュール内にプロシージャーを作成できます。モジュール内のプロシージャーの完全修飾名は、 <SCHEMA NAME>.<MODULE NAME>.<PROCEDURE NAME> となります。SQLExtendedProcedures() 関数および SQLExtendedProcedureColumns() 関数は、モジュールに関する情報を戻します。これらの関数は、現在の ODBC 仕様に定義されているものではありません。詳しくは、「*SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート*」の『モジュール』を参照してください。

構文

```
SQLRETURN   SQLExtendedProcedures    (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szProcCatalog */
               SQLSMALLINT       NameLength1,       /* cbProcCatalog */
               SQLCHAR           *SchemaName,       /* szProcSchema */
               SQLSMALLINT       NameLength2,       /* cbProcSchema */
               SQLCHAR           *ProcName,         /* szProcName */
               SQLSMALLINT       NameLength3),      /* cbProcName */
               SQLCHAR           *ProcModule,       /* szProcModule */
               SQLSMALLINT       NameLength4;       /* cbProcModule */
```

関数引数

表 55. SQLExtendedProcedures 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための *パターン値* が入れられるバッファー。 DB2 for MVS/ESA V 4.1 以降の場合、すべてのストアード・プロシージャーは 1 つのスキーマにあります。*SchemaName* 引数の受け入れ可能な値は NULL ポインターのみです。値を指定しても、空の結果セットと SQL_SUCCESS が戻されます。 DB2 Database for Linux, UNIX, and Windows 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。有効な検索パターンの詳細は、カタログ関数の入力引数の項を参照してください。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。

表 55. SQLExtendedProcedures 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLCHAR *	*ProcName*	入力	表名で結果セットを修飾するための *パターン値* を入れられるバッファー。
SQLSMALLINT	*NameLength3*	入力	*ProcName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ProcName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ProcModule*	入力	モジュール名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength4*	入力	*ProcModule* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ProcModule* がヌル終了ストリングの場合は SQL_NTS。

使用法

SQLExtendedProcedures() 関数から戻された結果セットには、所定の順序で SQLExtendedProcedures で戻される列にリストされている列が含まれています。これらの行は、PROCEDURE_CAT、PROCEDURE_SCHEMA、および PROCEDURE_NAME の順序になります。

多くの場合に SQLExtendedProcedures() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_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、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SQL_ATTR_LONGDATA_COMPAT 接続属性が設定されている場合、LOB 列タイプは LONG VARCHAR、LONG VARBINARY、または LONG VARGRAPHIC タイプとして報告されます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

ストアード・プロシージャーが DB2 for MVS/ESA V4.1 から V6 までの間のサーバーにある場合、そのストアード・プロシージャーの名前をサーバーの SYSIBM.SYSPROCEDURES カタログ表に登録しなければなりません。V8 およびそれ以降のサーバーの場合、そのサーバーの SYSIBM.SYSROUTINES および SYSIBM.SYSPARAMS カタログ表にストアード・プロシージャーを登録しなければなりません。

ストアード・プロシージャー・カタログの機能を提供しない DB2 サーバーの他のバージョンでは、空の結果セットが戻されます。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

SQLExtendedProcedures で戻される列

列 1 PROCEDURE_CAT (VARCHAR(128))
プロシージャー・カタログ名。このプロシージャーにカタログがない場合、この値は NULL になります。

列 2 PROCEDURE_SCHEM (VARCHAR(128))
PROCEDURE_NAME を含むスキーマの名前。

列 3 PROCEDURE_NAME (VARCHAR(128) NOT NULL)
プロシージャーの名前。

列 4 NUM_INPUT_PARAMS (INTEGER 非 NULL)
入力パラメーター数。INOUT パラメーターは、この数の中にカウントされません。

INOUT パラメーターの詳細を確かめるには、`SQLProcedureColumns()` から戻される COLUMN_TYPE 列を調べてください。

列 5 NUM_OUTPUT_PARAMS (INTEGER 非 NULL)
出力パラメーター数。INOUT パラメーターは、この数の中にカウントされません。

INOUT パラメーターの詳細を確かめるには、`SQLProcedureColumns()` から戻される COLUMN_TYPE 列を調べてください。

列 6 NUM_RESULT_SETS (INTEGER 非 NULL)
プロシージャーから戻される結果セット数。

この列は、今後 ODBC で使用するために予約済みになっているので、使用しないでください。

列 7 REMARKS (VARCHAR(254))
プロシージャーに関する記述情報が含まれています。

列 8 PROCEDURE_TYPE (SMALLINT)
プロシージャー・タイプを次のように定義します。

- SQL_PT_UNKNOWN: プロシージャーが値を戻すかどうかを判別することはできません。
- SQL_PT_PROCEDURE: 戻されるオブジェクトはプロシージャーで、戻り値がありません。
- SQL_PT_FUNCTION: 戻されるオブジェクトは関数で、戻り値があります。

CLI は常に SQL_PT_PROCEDURE を戻します。

列 9 SPECIFIC_NAME (VARCHAR(128))
PROCEDURE_NAME を固有に特定する名前。

列 10 PROCEDURE_MODULE (VARCHAR(128))
スキーマ内の PROCEDURE_NAME を含むモジュールの名前。

注:

- CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の SQLExtendedProcedures() 結果セットで定義されているものと同じです。
- 2 つのモジュールに同じ名前のプロシージャーが組み込まれている場合、SQLExtendedProcedures() 関数は両方のプロシージャーに関する詳細を戻します。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 56. SQLExtendedProcedures の SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。

表 56. SQLExtendedProcedures の SQLSTATE (続き)

SQLSTATE	説明	解説
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

ストアード・プロシージャー・カタログのサポートや、ストアード・プロシージャーのサポートを提供していない DB2 にアプリケーションを接続すると、SQLExtendedProcedures() 関数は空の結果セットを戻します。

SQLExtendedProcedureColumns 関数 (CLI) - プロシージャーの入出力パラメーター情報の取得

ストアード・プロシージャーに関連する入出力パラメーターのリストを戻します。

情報は SQL 結果セット内に戻されますが、照会により作成された結果セットを処理するのに使用される関数と同じ関数を使用して情報を取り出すことができます。

仕様:

- **CLI** 9.7

SQLExtendedProcedureColumns() は、ストアード・プロシージャーに関連する入出力パラメーターのリストを戻します。情報は SQL 結果セット内に戻されますが、照会により作成された結果セットを処理するのに使用される関数と同じ関数を使用して情報を取り出すことができます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLExtendedProcedureColumnsW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

モジュールとは、スキーマの概念を拡張したものです。 DB2 バージョン 9.7 以降のデータ・サーバーに接続するアプリケーションは、スキーマ内にモジュールを作成し、そのモジュール内にプロシージャーを作成できます。モジュール内のプロシージャーの完全修飾名は、 <SCHEMA NAME>.<MODULE NAME>.<PROCEDURE NAME> となります。SQLExtendedProcedures() 関数および SQLExtendedProcedureColumns() 関数は、モジュールに関する情報を戻します。これらの関数は、現在の ODBC 仕様に定義されているものではありません。詳しくは、「*SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート*」のを参照してください。

構文

```
SQLRETURN SQLExtendedProcedureColumns(
            SQLHSTMT        StatementHandle,   /* hstmt */
            SQLCHAR         *CatalogName,      /* szProcCatalog */
            SQLSMALLINT     NameLength1,       /* cbProcCatalog */
            SQLCHAR         *SchemaName,       /* szProcSchema */
            SQLSMALLINT     NameLength2,       /* cbProcSchema */
            SQLCHAR         *ProcName,         /* szProcName */
            SQLSMALLINT     NameLength3,       /* cbProcName */
            SQLCHAR         *ColumnName,       /* szColumnName */
            SQLSMALLINT     NameLength4),      /* cbColumnName */
            SQLCHAR         *ProcModule,       /* szProcModule */
            SQLSMALLINT     NameLength5;       /* cbProcModule */
```

関数引数

表 57. SQLExtendedProcedureColumns 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための *パターン値* が入れられるバッファー。 DB2 Database for Linux, UNIX, and Windows 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。有効な検索パターンの詳細は、カタログ関数の入力引数の項を参照してください。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ProcName*	入力	プロシージャー名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength3*	入力	*ProcName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ProcName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ColumnName*	入力	パラメーター名で結果セットを修飾するための *パターン値* が入れられるバッファー。この引数を使用するのは、 ProcName、SchemaName、またはその両方に空でない値を指定することによって既に限定されている結果セットをさらに修飾する場合です。

表 57. SQLExtendedProcedureColumns 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*NameLength4*	入力	*ColumnName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ColumnName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ProcModule*	入力	パラメーター名で結果セットを修飾するための *パターン値* が入れられるバッファー。この引数を使用するのは、 ProcName、SchemaName、または ColumnName に空でない値を指定することによって既に限定されている結果セットをさらに修飾する場合です。
SQLSMALLINT	*NameLength5*	入力	*ProcModule* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ProcModule* がヌル終了ストリングの場合は SQL_NTS。

使用法

SQLExtendedProcedureColumns() 関数は、PROCEDURE_CAT、PROCEDURE_SCHEM、PROCEDURE_NAME、COLUMN_TYPE、および PROCEDURE_MODULE の順序で結果セット内の情報を戻します。SQLExtendedProcedureColumns で戻される列は、結果セット内の列をリストしています。将来のリリースで最終列以降の列が定義される可能性があります。

多くの場合に SQLExtendedProcedureColumns() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() 関数を呼び出して、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、および COLUMN_NAME 列の実際の長さを判別することができます。

SQL_ATTR_LONGDATA_COMPAT 接続属性が設定されている場合、LOB 列タイプは LONG VARCHAR、LONG VARBINARY、または LONG VARGRAPHIC タイプとして報告されます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

ストアード・プロシージャー・カタログの機能を提供しない他の DB2 サーバーのバージョンでは、空の結果セットが戻されます。

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、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に関する記述子情報を戻すことはできません。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

SQLExtendedProcedureColumns で戻される列

列 1 PROCEDURE_CAT (VARCHAR(128))
プロシージャー・カタログの名前。このプロシージャーにカタログがない場合、この値は NULL になります。

列 2 PROCEDURE_SCHEM (VARCHAR(128))
PROCEDURE_NAME を含むスキーマの名前。

列 3 PROCEDURE_NAME (VARCHAR(128))
プロシージャーの名前。

列 4 COLUMN_NAME (VARCHAR(128))
パラメーターの名前。

列 5 COLUMN_TYPE (SMALLINT 非 NULL)
この行に関連した情報のタイプを識別します。値は次のとおりです。

- SQL_PARAM_TYPE_UNKNOWN : パラメーター・タイプが不明です。

  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。

列 6 DATA_TYPE (SMALLINT 非 NULL)
SQL データ・タイプ。

列 7 TYPE_NAME (VARCHAR(128) 非 NULL)
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。

列 8 COLUMN_SIZE (INTEGER)
SQL ルーチン内の XML 引数の場合、(XML 引数には長さがないため) ゼロが戻されます。しかし、カタログ式外部ルーチンの場合、XML パラメーターは XML AS CLOB(n) として宣言されます。この場合、COLUMN_SIZE はカタログされた長さ、n です。

DATA_TYPE 列の値が文字ストリングまたはバイナリー・ストリングを示す場合、この列には SQLCHAR または SQLWCHAR エレメント数で表記

した最大長が入れられます。DATA_TYPE 列の値が GRAPHIC (DBCS) ストリングの場合、COLUMN_SIZE はパラメーターの 2 バイトの SQLCHAR エレメントまたは SQLWCHAR エレメントの数になります。

日付、時刻、およびタイム・スタンプのデータ・タイプの場合、COLUMN_SIZE は文字データ・タイプに変換された場合に値を表示するために必要な SQLCHAR エレメントまたは SQLWCHAR エレメントの数の合計です。

数値データ・タイプの場合、これは結果セット内の NUM_PREC_RADIX 列の値に基づいて、列に許可されている総桁数または合計ビット数のいずれかです。

データ・タイプ精度の表を参照してください。

列 9 BUFFER_LENGTH (INTEGER)

SQL_C_DEFAULT が SQLBindCol()、SQLGetData() および SQLBindParameter() 呼び出しで指定された場合に、このパラメーターからのデータを保管するための関連する C バッファーの最大バイト。この長さには、NULL 終止符は含まれません。厳密な数データ・タイプを出すには、長さとして小数部や符号も考慮されます。

SQL ルーチン内の XML 引数の場合、(XML 引数には長さがないため) ゼロが戻されます。しかし、カタログ式外部ルーチンの場合、XML パラメーターは XML AS CLOB(n) として宣言されます。この場合、BUFFER_LENGTH はカタログされた長さ、n です。

データ・タイプ長の表を参照してください。

列 10 DECIMAL_DIGITS (SMALLINT)

パラメーターのスケール。スケールが適用できないデータ・タイプの場合は NULL が戻されます。

データ・タイプ・スケールの表を参照してください。

列 11 NUM_PREC_RADIX (SMALLINT)

10、2、または NULL のいずれか。DATA_TYPE が近似値データ・タイプである場合、この列には値 2 が含まれており、COLUMN_SIZE 列にはそのパラメーターに入れられるビット数が含まれています。

DATA_TYPE が高精度数値データ・タイプの場合、この列には値 10 が含まれており、COLUMN_SIZE 列と DECIMAL_DIGITS 列にはそのパラメーターに入れられる小数桁数が含まれています。

数値データ・タイプの場合、DBMS は 10 または 2 の NUM_PREC_RADIX を戻すことができます。

基数が適用できないデータ・タイプの場合は NULL が戻されます。

列 12 NULLABLE (SMALLINT 非 NULL)

パラメーターが NULL を受け入れない場合は SQL_NO_NULLS。

パラメーターが NULL 値を受け入れる場合は SQL_NULLABLE。

列 13 REMARKS (VARCHAR(254))

パラメーターに関する記述情報を入れられます。

列 14 COLUMN_DEF (VARCHAR)
列のデフォルト値。

NULL をデフォルト値として指定した場合、この列は引用符で囲まれていない語 NULL です。切り捨てを行わないとデフォルト値を表すことができない場合、この列には単一引用符で囲まれていない TRUNCATED が入ります。デフォルト値を指定しなかった場合、この列は NULL です。

COLUMN_DEF の値は、TRUNCATED 以外の値であれば新しい列定義を生成するために使用できます。

列 15 SQL_DATA_TYPE (SMALLINT 非 NULL)
SQL_DESC_TYPE 記述子のフィールドに表示される SQL データ・タイプの値。日時データ・タイプは除いて、この列は DATA_TYPE 列と同じです (CLI はインターバル・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 (SQL_CODE_DATE、SQL_CODE_TIME、または SQL_CODE_TIMESTAMP) のサブコードを戻します。

列 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 (CLI がサポートしていないインターバル・データ・タイプを含む)。

列 17 CHAR_OCTET_LENGTH (INTEGER)
文字データ・タイプ列のバイト単位の最大長。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。

列 18 ORDINAL_POSITION (INTEGER NOT NULL)
この結果セットの COLUMN_NAME で指定されているパラメーターの序数部が入れられます。ORDINAL_POSITION は、CALL ステートメント上で提供される引数の元の位置です。左端の引数の元の位置は、1 です。

列 19 IS_NULLABLE (Varchar)
- 列に NULL が含まれない場合は、NO。
- 列に NULL が含まれる場合は、YES。
- NULL 可能かどうか不明の場合は、ゼロ長ストリング。

NULL 可能かどうかを判別する際には、ISO 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。

列 20 SPECIFIC_NAME (VARCHAR(128))
PROCEDURE_NAME を固有に特定する名前。

列 21 PROCEDURE_MODULE (VARCHAR(128))
スキーマ内の PROCEDURE_NAME を含むモジュールの名前。

注:

- CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の SQLExtendedProcedureColumns() 結果セットで定義されているものと同じです。
- 2 つのモジュールに同じ名前のプロシージャーが組み込まれている場合、SQLExtendedProcedureColumns() 関数は両方のプロシージャーに関する詳細を戻します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 58. SQLExtendedProcedureColumns の SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
42601	PARMLIST 構文エラーです。	ストアード・プロシージャー・カタログ表の PARMLIST 値に、構文エラーがあります。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。

表 58. SQLExtendedProcedureColumns の SQLSTATE (続き)

SQLSTATE	説明	解説
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

SQLExtendedProcedureColumns() は、ストアード・プロシージャーから戻された可能性のある結果セットの属性についての情報を戻しません。

ストアード・プロシージャー・カタログのサポートや、ストアード・プロシージャーのサポートを提供していない DB2 サーバーにアプリケーションを接続すると、SQLExtendedProcedureColumns() は空の結果セットを戻します。

SQLExtendedProcedureColumns() は、現在のところ、 DB2 バージョン 9.7 以降でのみサポートされています。

例

```
/* get input/output parameter information for a procedure including
extended information */
cliRC = SQLExtendedProcedureColumns(hstmt,
 "CatalogName",
 SQL_NTS,
 "SchemaName",
 SQL_NTS,
 "ProcName",
 SQL_NTS,
 "ColumnName",
 SQL_NTS,
 "ModuleName",
 SQL_NTS );
```

SQLFetch function (CLI) - 次の行のフェッチ

結果セットの次の行へカーソルを進ませ、バインドされた列を取り出します。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

列を以下の位置にバインドすることができます。
- アプリケーション・ストレージ
- LOB ロケーター
- LOB ファイル参照

SQLFetch() を呼び出すと、適切なデータ転送が行われるとともに、列がバインドされたときに変換が指示されているとデータ変換が行われます。 SQLGetData() を呼び出して、フェッチの後に列を個々に受け取ることもできます。

SQLFetch() を呼び出せるのは、照会を実行するか、あるいは SQLGetTypeInfo() またはカタログ関数のどちらかの呼び出しによって結果セットが生成された (同一ステートメント・ハンドルを使用して) 後だけです。

構文

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

関数引数

表 59. SQLFetch 引数

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

使用法

SQLFetch() は、同じステートメント・ハンドルで結果セットが生成された後にのみ呼び出すことができます。 SQLFetch() を初めて呼び出す前には、カーソルを結果セットの先頭の前に置きます。

SQLBindCol() でバインドされたアプリケーション変数の個数が結果セット内の列数を超えてはなりません。超えると、SQLFetch() が失敗します。

SQLBindCol() を呼び出して列をバインドしなかった場合、 SQLFetch() はアプリケーションにデータを返さず、カーソルを先に進ませるだけです。この場合、SQLGetData() を呼び出して、すべての列を個々に取得することができます。カーソルが複数行カーソルの場合 (つまり、SQL_ATTR_ROW_ARRAY_SIZE が 1 より大きい場合)、 SQL_GETDATA_EXTENSIONS の *InfoType* を使用して SQLGetInfo() を呼び出すときに SQL_GD_BLOCK が返された場合にのみ、 SQLGetData() を呼び出すことができます。 (すべての DB2 データ・ソースが SQL_GD_BLOCK をサポートするわけではありません。) SQLFetch() でカーソルを次の行に進ませると、アンバインドされた列のデータが廃棄されます。固定長データ・タイプまたは小さい可変長データ・タイプの場合、 SQLGetData() を使用するよりも列をバインドしたほうがパフォーマンスが向上します。

LOB 値が大きすぎて 1 回の取り出しで検索できない場合、SQLGetData() (どの列タイプにも使用できる) を使用するか、または LOB ロケーターをバインドして SQLGetSubString() を使用することにより、LOB 値を部分単位で検索することができます。

バインドされたストレージ・バッファーが、 SQLFetch() から返されたデータを収容するのに十分な大きさでないと、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 が返され、切り捨てを示す SQLSTATE が生成されます。 SQLBindCol() 据え置き出力引数 *pcbValue* には、サーバーから取り出された列データの実際の長さが入っています。アプリケーションは、実際の出力の長さを入力バッファーの長さ (SQLBindCol() からの *pcbValue* 引数と *cbValueMax* 引数) と比較して、どの文字カラムが切り捨てられたかを判別する必要があります。

切り捨てが小数点の右側の桁数に関係している場合、数値データ・タイプの切り捨ては警告として報告されます。切り捨てが小数点の左側で行われると、エラーが返されます (診断のセクションを参照)。

GRAPHIC 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() で指定されている *cbValueMax* の値以下の最も大きい 2 バイトの倍数まで埋め込まれるという点だけが異なります。 CLI とアプリケーションとの間で転送された GRAPHIC (DBCS) データは、 C バッファー・タイプが SQL_C_CHAR のときにはヌル終了しません。(ただし、 CLI/ODBC 構成キーワード PATCH1 内で値 64 が使われていない場合に限ります)。バッファー・タイプが SQL_C_DBCHAR の場合は、GRAPHIC データはヌル終了します。

切り捨ては、SQL_ATTR_MAX_LENGTH ステートメント属性によっても影響されます。アプリケーションは、 SQL_ATTR_MAX_LENGTH および列ごとに返される最大長の値を指定して SQLSetStmtAttr() を呼び出し、同サイズ (に NULL 終止符文字分を加えたもの) の *rgbValue* バッファーを割り振ることによって、 CLI が切り捨てを報告しないように指定することができます。列データが設定された最大長より大きい場合、SQL_SUCCESS が返され、実際の長さではなく最大長が *pcbValue* に返されます。

結果セットからすべての行を取り出した場合や、残りの行が不要である場合、オプション SQL_CLOSE または SQL_DROP を指定した SQLCloseCursor() または SQLFreeStmt() を呼び出し、カーソルをクローズして残りのデータと関連リソースを廃棄する必要があります。

アプリケーションは、同じステートメント・ハンドルで SQLFetch() 呼び出しと SQLExtendedFetch() 呼び出しを混合できません。しかし、同じステートメント・ハンドルで SQLFetch() 呼び出しと SQLFetchScroll() 呼び出しを混合することはできます。SQLExtendedFetch() は使用すべきでないので、 SQLFetchScroll() に置き換えられていることに注意してください。

カーソルの位置決め

結果セットが作成されたら、カーソルは結果セットの先頭の前に置かれます。SQLFetch() は次の行セットをフェッチします。これは、*FetchOrientation* を SQL_FETCH_NEXT に設定して SQLFetchScroll() を呼び出すことと同等です。

SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性は、行セット内の行数を指定します。 SQLFetch() によってフェッチされている行セットが結果セットの最後とオーバーラップする場合、 SQLFetch() は行セットの一部を返します。つまり、S + R-1 が L より大きい場合、(この S はフェッチされている行セットの開始行、 R は行セット・サイズ、L は結果セットの最後の行)、行セットの最初の L-S+1 行のみが有効になります。残りの行は空であり、状況は SQL_ROW_NOROW になります。

詳細は、SQLFetchScroll() での SQL_FETCH_NEXT のカーソル位置決め規則を参照してください。

SQLFetch() が返された後では、現在行は行セットの最初の行になります。

行状況の配列

SQLFetch() は SQLFetchScroll() および SQLBulkOperations() と同じ方法で行状況配列の値を設定します。行状況配列は、行セットにある各行の状況を返すときに使用します。この配列のアドレスは、SQL_ATTR_ROW_STATUS_PTR ステートメント属性によって指定されます。

行フェッチ・バッファー

SQLFetch() は、データが返されなかった行も含め、行フェッチ・バッファー内にフェッチされた行の数を戻します。このバッファーのアドレスは、SQL_ATTR_ROWSFETCHED_PTR ステートメント属性によって指定されます。このバッファーは、SQLFetch() と SQLFetchScroll() によって設定されます。

エラー処理

エラーと警告は、個々の行、または関数全体に対して出されます。エラーと警告は、SQLGetDiagField() 関数を使って取り出すことができます。

関数全体についてのエラーおよび警告

エラーが、関数全体に適用される場合、例えば SQLSTATE HYT00 (タイムアウト満了)、または SQLSTATE 24000 (カーソル状態が無効) の場合、SQLFetch() は SQL_ERROR と、適用可能な SQLSTATE を返します。行セット・バッファーの内容は定義されず、カーソル位置は変更されません。

警告が関数全体に当てはまる場合、 SQLFetch() は SQL_SUCCESS_WITH_INFO と適用可能な SQLSTATE を返します。関数全体に適用される警告の状況レコードは、個々の行に適用される状況レコードよりも前に返されます。

個々の行のエラーおよび警告

エラー (SQLSTATE 22012 (ゼロによる除算) または警告 SQLSTATE 01004 (データが切り捨てられた)) が単一行に適用される場合、どの行でもエラーが生じた (その場合は SQL_ERROR が戻されます) のでないかぎり SQLFetch() は SQL_SUCCESS_WITH_INFO を戻します。 SQLFetch() は以下も行います。

- エラーの場合は、行状況配列の対応するエレメントを SQL_ROW_ERROR に設定し、警告の場合は、SQL_ROW_SUCCESS_WITH_INFO に設定します。
- エラーまたは警告用の SQLSTATE を含むゼロ個以上の状況レコードを追加します。
- 状況レコードの行または列番号フィールドを設定します。 SQLFetch() が行または列番号を判別できない場合は、その番号を SQL_ROW_NUMBER_UNKNOWN または SQL_COLUMN_NUMBER_UNKNOWN に設定します。状況レコードが特定の列に当てはまらない場合、SQLFetch() は列番号を SQL_NO_COLUMN_NUMBER に設定します。

SQLFetch() は行番号の順に状況レコードを返します。つまり、認識されていない行 (存在する場合) の状況レコードをすべて返してから、最初の行 (存在する場合) の状況レコードをすべて返し、その後、 2 番目の行 (存在する場合) の状況レコードを返す、という具合に続きます。それぞれの行の状況レコードは、通常の状況レコードの配列規則に基づいて配列されます。

記述子と SQLFetch

以下のセクションでは、SQLFetch() が記述子と対話する方法について説明します。

引数のマッピング

ドライバーは、引数 SQLFetch() に基づいて記述子フィールドを設定することはありません。

その他の記述子フィールド

以下の記述子フィールドは、SQLFetch() によって使用されます。

表 60. 記述子フィールド

記述子フィールド	説明	ロケーション	設定時に使用するもの
SQL_DESC_ARRAY_SIZE	ARD	ヘッダー	SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性
SQL_DESC_ARRAY_STATUS_PTR	IRD	ヘッダー	SQL_ATTR_ROW_STATUS_PTR ステートメント属性
SQL_DESC_BIND_OFFSET_PTR	ARD	ヘッダー	SQL_ATTR_ROW_BIND_OFFSET_PTR ステートメント属性
SQL_DESC_BIND_TYPE	ARD	ヘッダー	SQL_ATTR_ROW_BIND_TYPE ステートメント属性
SQL_DESC_COUNT	ARD	ヘッダー	SQLBindCol() の *ColumnNumber* 引数
SQL_DESC_DATA_PTR	ARD	レコード	SQLBindCol() の *TargetValuePtr* 引数
SQL_DESC_INDICATOR_PTR	ARD	レコード	SQLBindCol() の *StrLen_or_IndPtr* 引数
SQL_DESC_OCTET_LENGTH	ARD	レコード	SQLBindCol() の *BufferLength* 引数
SQL_DESC_OCTET_LENGTH_PTR	ARD	レコード	SQLBindCol() の *StrLen_or_IndPtr* 引数
SQL_DESC_ROWS_PROCESSED_PTR	IRD	ヘッダー	SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性
SQL_DESC_TYPE	ARD	レコード	SQLBindCol() の *TargetType* 引数

すべての記述子フィールドは SQLSetDescField() を介して設定することもできます。

長さおよび標識バッファーの分離

アプリケーションは、単一のバッファーまたは 2 つの別個のバッファーをバインドして、長さおよび標識値を保持することができます。アプリケーションが SQLBindCol() を呼び出すときに、 ARD の SQL_DESC_OCTET_LENGTH_PTR と SQL_DESC_INDICATOR_PTR フィールドは同一のアドレスに設定されます。これは、*StrLen_or_IndPtr* 引数に渡されます。アプリケーションが SQLSetDescField() または SQLSetDescRec() を呼び出すときに、これらの 2 つのフィールドを別々のアドレスに設定することができます。

SQLFetch() は、アプリケーションが別々の長さおよび標識バッファーを指定したかどうかを判別します。この場合にデータが NULL でないなら、SQLFetch() は標識バッファーを 0 に設定し、長さバッファーに長さを返します。データが NULL の

場合、SQLFetch() は標識バッファーを SQL_NULL_DATA に設定し、長さバッファーは修正しません。

戻りコード

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

結果セット内に行がない場合、または直前の SQLFetch() 呼び出しで結果セットからすべての行がフェッチされた場合、 SQL_NO_DATA_FOUND が返されます。

すべての行がフェッチされた場合、カーソルは結果セットの末尾の後ろに置かれます。

診断

表 61. SQLFetch SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	1 つ以上の列について返されたデータが切り捨てられました。ストリング値または数値は右方切り捨てされます。 (エラーが発生しなかった場合、SQL_SUCCESS_WITH_INFO が返されます。)
07002	列が多すぎます。	バインド中に指定された 1 つ以上の列の列番号が、結果セット内の列数より大きい値でした。
07006	無効な変換です。	データ値を、 SQLBindCol() の *fCType* で指定されたデータ・タイプに有意義な方法で変換できませんでした。
07009	無効な記述子索引	列 0 がバインドされましたが、ブックマークが使用されませんでした (SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF に設定されました)。
22002	無効な出力または標識バッファーが指定されました。	SQLBindCol() の引数 *pcbValue* に指定されたポインター値は NULL ポインターで、対応する列の値は NULL です。 SQL_NULL_DATA を報告する手段はありません。 SQLBindFileToCol() の引数 IndicatorValue に指定されたポインターは NULL ポインターで、対応する LOB 列は NULL です。 SQL_NULL_DATA を報告する手段はありません。
22003	数値が範囲外です。	1 つ以上の列の数値を (数値またはストリングとして) 返したため、割り当て時または中間結果の計算時に数値の整数部分が切り捨てられたと考えられます。 ゼロでの除算が行われた算術式からの値が返されました。 **注:** このエラーが DB2 Database for Linux, UNIX, and Windows によって検出される場合、関連したカーソルは未定義になります。エラーが CLI または他の IBM RDBMS によって検出された場合、カーソルはオープンされたままとなり、後続のフェッチ呼び出しに進みます。

表 61. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説
22005	割り当てにエラーがありました。	返された値は、バインドのデータ・タイプと互換性がありませんでした。 返された LOB ロケーター値は、バインドされた列のデータ・タイプと互換性がありませんでした。
22007	無効な日付時刻形式です。	文字ストリングから日時フォーマットへの変換が指定されましたが、無効なストリング表示または値が指定されたか、あるいは値が無効な日付になっています。 日付、時刻、またはタイム・スタンプの値が、指定された日付タイプの構文に従っていません。
22008	日時フィールドがオーバーフローしました。	日時フィールドのオーバーフローが発生しました。例えば、日付またはタイム・スタンプの算術計算の結果が有効な日付範囲内の値にならないか、またはバインドされた変数が小さすぎるため日時値を代入できません。
22012	0 による除算は無効です。	ゼロでの除算が行われた算術式からの値が返されました。
24000	カーソル状態が無効です。	ステートメント・ハンドルで実行された直前の SQL ステートメントは照会ではありませんでした。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。

428A1	ホスト・ファイル変数によって参照されたファイルにアクセスできません。	これは、以下のシナリオで生ずる可能性があります。テキスト内で関連付けられた理由コードは、特定のエラーを表します。
		• 01 - ファイル名の長さが無効か、またはファイル名とパスの片方または両方のフォーマットが無効です。
		• 02 - ファイル・オプションが無効です。ファイル・オプションには、以下のいずれかの値が指定されなければなりません。
		`SQL_FILE_READ -既存ファイルからの読み取り`
		`SQL_FILE_CREATE -書き込みのための新規ファイルの作成`
		`SQL_FILE_OVERWRITE -既存ファイルの上書き ファイルが存在しない場合は ファイルを作成`
		`SQL_FILE_APPEND -既存ファイルへの付加 ファイルが存在しない場合は ファイルを作成`
		• 03 - ファイルが見つかりませんでした。
		• 04 - SQL_FILE_CREATE オプションが、既存のファイルと同じ名前を持つファイルに指定されました。
		• 05 - ファイルへのアクセスが拒否されました。ユーザーが、ファイルをオープンするための権限を持っていません。
		• 06 - ファイルへのアクセスが拒否されました。ファイルが非互換モードで使用中です。書き込まれるファイルが、排他モードでオープンされています。
		• 07 - ファイルへの書き込み中に、ディスクがフルになりました。
		• 08 - ファイルの読み取り中に、想定外のファイル終わりが見つかりました。
		• 09 - ファイルのアクセス中に、メディア・エラーが起きました。

表 61. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説
54028	並行 LOB ハンドルが最大数に達しました。	割り当てられた最大 LOB ロケーター。 並行 LOB ロケーターの最大数に達しました。新しいロケーターを割り当てることができません。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	SQLExtendedFetch() を呼び出した後、 SQL_CLOSE オプションを指定して SQLFreeStmt() を呼び出す前に、 StatementHandle の SQLFetch() を呼び出しました。 SQLPrepare() または SQLExecDirect() を *StatementHandle* 用に呼び出す前に、この関数が呼び出されました。 実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY092	オプション・タイプが範囲外です。	直前の SQLBindFileToCol() 操作の *FileOptions* 引数が無効でした。
HYC00	ドライバーが使用できません。	CLI またはデータ・ソースは、 SQLBindCol() または SQLBindFileToCol() の *fCType* および対応する列の SQL データ・タイプの組み合わせによって指定された変換をサポートしません。 CLI によってサポートされていない列データ・タイプに対して、SQLBindCol() 呼び出しが行われました。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

例

```
/* fetch each row and display */
cliRC = SQLFetch(hstmt);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

if (cliRC == SQL_NO_DATA_FOUND)
{
  printf("¥n  Data not found.¥n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
  printf("

  /* fetch next row */
  cliRC = SQLFetch(hstmt);
  STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
}
```

SQLFetchScroll 関数 (CLI) - 行セットをフェッチし、すべてのバインドされた列のデータを戻す

結果セットからデータの指定された行セットをフェッチし、すべてのバインドされた列のデータを戻します。

行セットは、絶対位置または相対位置で指定するか、またはブックマークを使用して指定できます。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

構文

```
SQLRETURN   SQLFetchScroll   (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       FetchOrientation,
                              SQLLEN            FetchOffset);
```

関数引数

表 62. SQLFetchScroll 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*FetchOrientation*	入力	フェッチのタイプ。以下のいずれかです。 • SQL_FETCH_NEXT • SQL_FETCH_PRIOR • SQL_FETCH_FIRST • SQL_FETCH_LAST • SQL_FETCH_ABSOLUTE • SQL_FETCH_RELATIVE • SQL_FETCH_BOOKMARK 詳しくは、カーソルの位置決めを参照してください。

表 62. SQLFetchScroll 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLLEN	*FetchOffset*	入力	フェッチを行う行の数。この引数の解釈は、*FetchOrientation* 引数の値によって異なります。詳しくは、カーソルの位置決めを参照してください。

使用法

概説

SQLFetchScroll() は、結果セットから指定した行セットを返します。行セットは、絶対位置または相対位置で指定するか、またはブックマークを使用して指定できます。 SQLFetchScroll() を呼び出すことができるのは、結果セットが存在するときだけです。つまり、結果セットを作成する呼び出しを行ってから、その結果セット上にあるカーソルをクローズするまでの間です。列のいずれかがバインドされている場合、これらの列のデータが返されます。アプリケーションが行状況配列にポインターを指定するか、またはフェッチされている行の数を返す先のバッファーを指定している場合、 SQLFetchScroll() はこの情報も共に返します。 SQLFetchScroll() の呼び出しは、 SQLFetch() の呼び出しと混合できますが、 SQLExtendedFetch() の呼び出しと混合することはできません。

カーソルの位置決め

結果セットが作成されたら、カーソルは結果セットの先頭の前に置かれます。SQLFetchScroll() は、次の表に示されているように、 *FetchOrientation* と *FetchOffset* 引数の値に基づいてブロック・カーソルの位置を決めます。新しい行セットの開始を決定するための正確な規則は、次のセクションで示されています。

FetchOrientation
意味

SQL_FETCH_NEXT
次の行セットを返します。これは、SQLFetch() の呼び出しと同等です。SQLFetchScroll() は、*FetchOffset* の値を無視します。

SQL_FETCH_PRIOR
直前の行セットを返します。 SQLFetchScroll() は、*FetchOffset* の値を無視します。

SQL_FETCH_RELATIVE
行セット *FetchOffset* を現在の行セットの開始から返します。

SQL_FETCH_ABSOLUTE
行 *FetchOffset* から始まる行セットを返します。

SQL_FETCH_FIRST
結果セットにある最初の行セットを返します。 SQLFetchScroll() は、*FetchOffset* の値を無視します。

SQL_FETCH_LAST
結果セットにある最後に完了した行セットを返します。 SQLFetchScroll() は、*FetchOffset* の値を無視します。

SQL_FETCH_BOOKMARK
SQL_ATTR_FETCH_BOOKMARK_PTR ステートメント属性によって指定したブックマークから、行セット *FetchOffset* の行を返します。

すべてのカーソルが、これらのオプションをすべてサポートするわけではありません。例えば静的前方スクロール・カーソルは、SQL_FETCH_NEXT だけをサポートします。またキー・セット・カーソルなどの両方向スクロール・カーソルは、これらのオプションをすべてサポートします。 SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性は、行セット内の行数を指定します。 SQLFetchScroll() によってフェッチされている行セットが結果セットの最後とオーバーラップする場合は、SQLFetchScroll() は行セットの一部を返します。つまり、S + R-1 が L より大きい場合、(この S はフェッチされている行セットの開始行、 R は行セット・サイズ、L は結果セットの最後の行)、行セットの最初の L-S+1 行のみが有効になります。残りの行は空であり、状況は SQL_ROW_NOROW になります。

SQLFetchScroll() が返ったら、行セット・カーソルは結果セットの最初の行に置かれます。

バインドされた列のデータを返す

SQLFetchScroll() は、SQLFetch() と同様に、バインドされた列のデータを返します。

バインドされた列がない場合、SQLFetchScroll() はデータを返しませんが、ブロック・カーソルを指定した位置にまで移動します。この場合、SQLFetch() のときと同じように SQLGetData() を使用して情報を検索できます。

行状況の配列

行状況配列は、行セットにある各行の状況を返すときに使用します。この配列のアドレスは、SQL_ATTR_ROW_STATUS_PTR ステートメント属性によって指定されます。配列は、アプリケーションによって割り当てられており、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性によって指定されている数と同じ数のエレメントがあるべきです。その値は、SQLFetch()、SQLFetchScroll()、SQLSetPos() によって設定されています (カーソルが SQLExtendedFetch() によって配置された後にその値が呼び出された場合は例外です)。SQL_ATTR_ROW_STATUS_PTR ステートメント属性の値が NULL ポインターである場合、これらの関数は行状況を返しません。

行状況配列バッファーの内容は、SQLFetch() または SQLFetchScroll() が SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さない場合には定義されません。

以下の値が行状況配列に返されます。

行状況配列の値
説明

SQL_ROW_SUCCESS
行が正常にフェッチされました。

SQL_ROW_SUCCESS_WITH_INFO
行が正常にフェッチされました。しかし、行に関する警告が返されました。

SQL_ROW_ERROR
行をフェッチ中にエラーが発生しました。

SQL_ROW_ADDED
この行は SQLBulkOperations() に挿入されました。この行がもう一度フェッチされたり、SQLSetPos() によって更新されたりすると、状況は SQL_ROW_SUCCESS になります。

この値は、SQLFetch() または SQLFetchScroll() によっては設定されません。

SQL_ROW_UPDATED
行は正常にフェッチされ、この結果セットからフェッチされた最後のフェッチ以降に更新されています。この行がこの結果セットからもう一度フェッチされたり、 SQLSetPos() によって更新されたりすると、その行の状況は新しい状況に変更されます。

SQL_ROW_DELETED
この行は、この結果セットからの最後のフェッチ以降に削除されています。

SQL_ROW_NOROW
行セットが結果セットの終了行と重なり合いました。行状況配列のこのエレメントに対応した行が返されていません。

行フェッチ・バッファー

行フェッチ・バッファーは、フェッチした行の数を返すために使用します。これには、フェッチを行っているときにエラーが生じたためにデータが返されなかった行も含まれます。言い換えると、行状況配列の値が SQL_ROW_NOROW ではない行の数ということになります。このバッファーのアドレスは、SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性によって指定されます。バッファーは、アプリケーションによって割り当てられます。これは SQLFetch() と SQLFetchScroll() によって設定されます。 SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性の値が NULL ポインターである場合、これらの関数はフェッチした行の数を返しません。結果セットの現在行の数を判別するために、アプリケーションは SQL_ATTR_ROW_NUMBER 属性を使用して SQLGetStmtAttr() を呼び出すことができます。

行フェッチ・バッファーの内容は、SQLFetch() または SQLFetchScroll() が SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さない場合には定義されません。ただし、SQL_NO_DATA が返された場合は例外です。この場合は、行フェッチ・バッファーの値は 0 に設定されます。

エラー処理

SQLFetchScroll() は SQLFetch() と同じ方法でエラーと警告を返します。

記述子および SQLFetchScroll()

SQLFetchScroll() は、SQLFetch() と同じ方法で記述子と対話します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

- SQL_NO_DATA
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

各 SQLSTATE 値に関連している戻りコードは、特に注記がない限り SQL_ERROR です。エラーが単一の列に生じる場合、 SQL_DIAG_COLUMN_NUMBER の *DiagIdentifier* を使用して SQLGetDiagField() を呼び出し、エラーが生じた列を判別できます。また、SQL_DIAG_ROW_NUMBER の *DiagIdentifier* を使用して SQLGetDiagField() を呼び出し、その列を含む行を判別できます。

表 63. SQLFetchScroll SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	非ブランク文字または非 NULL のバイナリー・データを切り捨てた結果として、ストリングまたはバイナリー・データが列に返されました。ストリング値は右方切り捨てされます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S01	行にエラーがあります。	1 つ以上の行をフェッチ中にエラーが起きました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。) (この SQLSTATE は、CLI v2 に接続したときのみに返されます。)
01S06	結果セットが最初の行セットを戻す前に、取り出しを試行しました。	要求された行セットは、現行位置が最初の行を超えたときに結果セットの開始と重なり合いました。そして、*FetchOrientation* が SQL_PRIOR であるか、または *FetchOrientation* が SQL_RELATIVE でした。これは、絶対値が現行の SQL_ATTR_ROW_ARRAY_SIZE 以下である負の *FetchOffset* を伴います。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S07	小数点以下切り捨てです。	列に返されたデータが切り捨てられました。数値データ・タイプの場合、数の小数部分は切り捨てられます。時刻またはタイム・スタンプのデータ・タイプの場合、時刻の小数部分は切り捨てられました。
07002	列が多すぎます。	バインド中に指定された 1 つ以上の列の列番号が、結果セット内の列数より大きい値でした。
07006	無効な変換です。	結果セットにある列のデータ値を、 SQLBindCol() の *TargetType* で指定された C データ・タイプに変換できませんでした。
07009	記述子索引が無効です。	列 0 がバインドされ、SQL_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF に設定されました。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
22001	ストリング・データの右側が切り捨てられました。	行に返された可変長ブックマークが切り捨てられました。
22002	無効な出力または標識バッファーが指定されました。	NULL データがフェッチされ、SQLBindCol() によって設定された *StrLen_or_IndPtr* (あるいは、SQLSetDescField() または SQLSetDescRec() によって設定された SQL_DESC_INDICATOR_PTR) が NULL ポインターである列に入れられました。)

表 63. SQLFetchScroll SQLSTATE (続き)

SQLSTATE	説明	解説
22003	数値が範囲外です。	1 つ以上のバインド済み列の数値を (数値またはストリングとして) 返したため、数値の整数部分 (小数部分ではなく) が切り捨てられたと考えられます。
22007	無効な日付時刻形式です。	結果セットにある文字カラムが日付、時刻、またはタイム・スタンプ C 構造にバインドされ、列にある値は無効である日付、時刻、またはタイム・スタンプにバインドされました。
22012	0 による除算は無効です。	ゼロでの除算が行われた算術式からの値が返されました。
22018	キャスト指定の文字値が無効です。	結果セットにある文字カラムが文字 C バッファーにバインドされ、その列には、バッファーの文字セットに表示のない文字が入っていました。結果セットにある文字カラムが近似の数値 C バッファーにバインドされ、その列にある値は有効で近似の数値にキャストできませんでした。結果セットにある文字カラムが厳密な数 C バッファーにバインドされ、その列にある値は有効で厳密な数にキャストできませんでした。結果セットにある文字カラムは日時 C バッファーにバインドされ、その列にある値を有効な日時値にキャストできませんでした。
24000	カーソル状態が無効です。	*StatementHandle* は実行状態にありましたが、 *StatementHandle* に関連する結果セットがありませんでした。
40001	トランザクションのロールバック	フェッチが実行されたトランザクションは、デッドロックを避けるために終了しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 `SQLCancel()` がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。

表 63. SQLFetchScroll SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	指定した *StatementHandle* が実行状態にありませんでした。関数は、最初に SQLExecDirect()、SQLExecute()、またはカタログ関数を呼び出さずに呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。 SQLExtendedFetch() を呼び出した後、SQL_CLOSE オプションを指定して SQLFreeStmt() を呼び出す前に、*StatementHandle* で SQLFetchScroll() を呼び出しました。
HY106	フェッチ・タイプが範囲外です。	引数 *FetchOrientation* に指定された値は無効でした。 引数 *FetchOrientation* が SQL_FETCH_BOOKMARK であり、SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF に設定されました。 SQL_CURSOR_TYPE ステートメント属性の値は SQL_CURSOR_FORWARD_ONLY であり、引数 *FetchOrientation* の値は SQL_FETCH_NEXT ではありませんでした。
HY107	行の値が範囲外です。	SQL_ATTR_CURSOR_TYPE ステートメント属性で指定した値は SQL_CURSOR_KEYSET_DRIVEN でしたが、SQL_ATTR_KEYSET_SIZE ステートメント属性で指定した値は 0 より大きく、 SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性で指定した値より小さいものでした。
HY111	ブックマークの値が無効です。	引数 *FetchOrientation* は SQL_FETCH_BOOKMARK であり、SQL_ATTR_FETCH_BOOKMARK_PTR ステートメント属性で値によって指されているブックマークは無効であるかまたは NULL ポインターでした。
HYC00	ドライバーが使用できません。	指定されたフェッチ・タイプは、サポートされません。 SQLBindCol() の *TargetType* と、対応する列の SQL データ・タイプの組み合わせによって指定されている変換はサポートされていません。

制限

なし。

例

```
/* fetch the rowset: row15, row16, row17, row18, row19 */
printf("¥n  Fetch the rowset: row15, row16, row17, row18, row19.¥n");

/* fetch the rowset and return data for all bound columns */
cliRC = SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 15);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
```

```
/* call SQLFetchScroll with SQL_FETCH_RELATIVE offset 3 */
printf("  SQLFetchScroll with SQL_FETCH_RELATIVE offset 3.¥n");
printf("    COL1          COL2         ¥n");
printf("    ------------  -------------¥n");

/* fetch the rowset and return data for all bound columns */
cliRC = SQLFetchScroll(hstmt, SQL_FETCH_RELATIVE, 3);
```

SQLFetchScroll() (CLI) のカーソル位置決めの規則

次のセクションでは、*FetchOrientation* の各値の正確な規則について説明します。これらの規則には、以下の表記を使用します。

FetchOrientation
意味

開始前 ブロック・カーソルが結果セットの先頭の前に置かれています。新しい行セットの最初の行が結果セットの先頭の前に置かれている場合、SQLFetchScroll() は SQL_NO_DATA を返します。

終了後 ブロック・カーソルが結果セットの末尾の後ろに置かれています。新しい行セットの最初の行が結果セットの末尾の後ろに置かれている場合、SQLFetchScroll() は SQL_NO_DATA を返します。

CurrRowsetStart
現在の行セット内の最初の行の番号。

LastResultRow
結果セットにある最後の行の番号。

RowsetSize
行セットのサイズ。

FetchOffset
FetchOffset 引数の値。

BookmarkRow
SQL_ATTR_FETCH_BOOKMARK_PTR ステートメント属性によって指定したブックマークに対応する行。

SQL_FETCH_NEXT 規則は以下のとおりです。

表 64. SQL_FETCH_NEXT 規則

条件	新しい行セットの最初の行
開始前	1
CurrRowsetStart + RowsetSize <= LastResultRow	CurrRowsetStart + RowsetSize
CurrRowsetStart + RowsetSize > LastResultRow	終了後
終了後	終了後

SQL_FETCH_PRIOR 規則は以下のとおりです。

表 65. SQL_FETCH_PRIOR 規則

条件	新しい行セットの最初の行
開始前	開始前

表 65. SQL_FETCH_PRIOR 規則 (続き)

条件	新しい行セットの最初の行
CurrRowsetStart = 1	開始前
1 < CurrRowsetStart <= RowsetSize	1 [a]
CurrRowsetStart > RowsetSize	CurrRowsetStart - RowsetSize
終了後でしかも LastResultRow < RowsetSize	1 [a]
終了後でしかも LastResultRow >= RowsetSize	LastResultRow - RowsetSize + 1

- **a** SQLFetchScroll() は SQLSTATE 01S06 (結果セットが最初の行セットを戻す前に、取り出しを試行しました。) と SQL_SUCCESS_WITH_INFO を返します。

SQL_FETCH_RELATIVE 規則は以下のとおりです。

表 66. SQL_FETCH_RELATIVE 規則

条件	新しい行セットの最初の行
(開始前でしかも *FetchOffset* > 0) または (終了後でしかも *FetchOffset* 0)	-- [a]
開始前でしかも *FetchOffset* <= 0	開始前
CurrRowsetStart = 1 で、しかも FetchOffset < 0	開始前
CurrRowsetStart > 1 で、しかも CurrRowsetStart + *FetchOffset* < 1、しかも |*FetchOffset*| > RowsetSize	開始前
CurrRowsetStart > 1 で、しかも CurrRowsetStart + *FetchOffset* < 1、しかも |*FetchOffset*| < RowsetSize	1 [b]
1 <= CurrRowsetStart + *FetchOffset* <= LastResultRow	CurrRowsetStart + *FetchOffset*
CurrRowsetStart + *FetchOffset* > LastResultRow	終了後
終了後でしかも *FetchOffset* >= 0	終了後

- **a** SQLFetchScroll() は、 *FetchOrientation* セットを使用して呼び出されたときと同じ行セットを SQL_FETCH_ABSOLUTE に返します。詳細については、「SQL_FETCH_ABSOLUTE」の項を参照してください。
- **b** SQLFetchScroll() は SQLSTATE 01S06 (結果セットが最初の行セットを戻す前に、取り出しを試行しました。) と SQL_SUCCESS_WITH_INFO を返します。

SQL_FETCH_ABSOLUTE 規則は以下のとおりです。

表 67. SQL_FETCH_ABSOLUTE 規則

条件	新しい行セットの最初の行
FetchOffset < 0 でしかも |*FetchOffset*| <= LastResultRow	LastResultRow + *FetchOffset* + 1
FetchOffset < 0 でしかも |*FetchOffset*| > LastResultRow でしかも |FetchOffset| > RowsetSize	開始前
FetchOffset < 0 でしかも |*FetchOffset*| > LastResultRow でしかも |FetchOffset| <= RowsetSize	1 [a]
FetchOffset = 0	開始前
1 <= *FetchOffset* <= LastResultRow	*FetchOffset*

表 67. SQL_FETCH_ABSOLUTE 規則 (続き)

条件	新しい行セットの最初の行
FetchOffset > LastResultRow	終了後

- **a** SQLFetchScroll() は SQLSTATE 01S06 (結果セットが最初の行セットを戻す前に、取り出しを試行しました。) と SQL_SUCCESS_WITH_INFO を返します。

SQL_FETCH_FIRST 規則は以下のとおりです。

表 68. SQL_FETCH_FIRST 規則

条件	新しい行セットの最初の行
任意	1

SQL_FETCH_LAST 規則は以下のとおりです。

表 69. SQL_FETCH_LAST 規則

条件	新しい行セットの最初の行
RowsetSize = LastResultRow	LastResultRow - RowsetSize + 1
RowsetSize > LastResultRow	1

SQL_FETCH_BOOKMARK 規則は以下のとおりです。

表 70. SQL_FETCH_BOOKMARK 規則

条件	新しい行セットの最初の行
BookmarkRow + *FetchOffset* <1	開始前
1 <= BookmarkRow + *FetchOffset* <= LastResultRow	BookmarkRow +*FetchOffset*
BookmarkRow + *FetchOffset* > LastResultRow	終了後

SQLForeignKeys 関数 (CLI) - 外部キー列のリストの取得

指定された表の外部キーに関する情報を返します。

情報は SQL 結果セット内に返されますが、これは、照会で生成された結果を検索するのに使用される関数と同じ関数を使用して処理することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

SQLForeignKeys() 関数は、指定された表の外部キーに関する情報を返します。この情報は SQL 結果セットで返されます。この結果セットは、照会によって生成される結果を取得するために使用するのと同じ関数を使用して処理できます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLForeignKeysW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLForeignKeys   (
                SQLHSTMT         StatementHandle,  /* hstmt */
                SQLCHAR          *PKCatalogName,   /* szPkCatalogName */
                SQLSMALLINT      NameLength1,      /* cbPkCatalogName */
                SQLCHAR          *PKSchemaName,    /* szPkSchemaName */
                SQLSMALLINT      NameLength2,      /* cbPkSchemaName */
                SQLCHAR          *PKTableName,     /* szPkTableName */
                SQLSMALLINT      NameLength3,      /* cbPkTableName */
                SQLCHAR          *FKCatalogName,   /* szFkCatalogName */
                SQLSMALLINT      NameLength4,      /* cbFkCatalogName */
                SQLCHAR          *FKSchemaName,    /* szFkSchemaName */
                SQLSMALLINT      NameLength5,      /* cbFkSchemaName */
                SQLCHAR          *FKTableName,     /* szFkTableName */
                SQLSMALLINT      NameLength6);     /* cbFkTableName */
```

関数引数

表 71. SQLForeignKeys 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*PKCatalogName*	入力	3 つの部分から成る主キー表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *PKCatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*PKCatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *PKCatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*PKSchemaName*	入力	主キー表のスキーマ修飾子。
SQLSMALLINT	*NameLength2*	入力	*PKSchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *PKSchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*PKTableName*	入力	主キーを収めた表名の名前。
SQLSMALLINT	*NameLength3*	入力	*PKTableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *PKTableName* がヌル終了ストリングの場合は SQL_NTS。

表 71. SQLForeignKeys 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLCHAR *	*FKCatalogName*	入力	3 つの部分から成る外部キー表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *FKCatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength4*	入力	*FKCatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *FKCatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*FKSchemaName*	入力	外部キーを含む表のスキーマ修飾子。
SQLSMALLINT	*NameLength5*	入力	*FKSchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *FKSchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*FKTableName*	入力	外部キーを含む表の名前。
SQLSMALLINT	*NameLength6*	入力	*FKTableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *FKTableName* がヌル終了ストリングの場合は SQL_NTS。

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() 関数は指定された表の主キーとその表を参照するすべての外部キー (他の表の中にある) を含む結果セットを返します。

FKTableName に表名が入っていて、*PKTableName* が空ストリングである場合、SQLForeignKeys() 関数は指定された表のすべての外部キーとそれらのキーが参照する主キー (他の表の中にある) を含む結果セットを返します。

PKTableName と *FKTableName* の両方に表名が入っている場合、 SQLForeignKeys() 関数は *FKTableName* で指定された表の外部キーを返しますが、これらの外部キーは *PKTableName* で指定された表の主キーを参照します。キーは最大 1 つでなければなりません。

表名に関連したスキーマ修飾子引数を指定しないと、スキーマ名は現行の接続にとって現在有効である表名にデフォルト設定されます。

SQLForeignKeys で戻される列には、SQLForeignKeys() 呼び出しで生成された結果セットの列をリストしています。主キーに関連した外部キーを要求すると、結果セットは FKTABLE_CAT、FKTABLE_SCHEM、FKTABLE_NAME、そして

ORDINAL_POSITION の順序になります。外部キーに関連した主キーが要求されると、結果セットは PKTABLE_CAT、PKTABLE_SCHEM、PKTABLE_NAME、そして ORDINAL_POSITION の順序になります。

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_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、接続先の DBMS でサポートされている関連の TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLForeignKeys で戻される列

列 1 PKTABLE_CAT (VARCHAR(128))
PKTABLE_NAME のカタログの名前。この表にカタログがない場合、この値は NULL になります。

列 2 PKTABLE_SCHEM (VARCHAR(128))
PKTABLE_NAME を収めたスキーマの名前。

列 3 PKTABLE_NAME (VARCHAR(128) 非 NULL)
主キーを収めた表の名前。

列 4 PKCOLUMN_NAME (VARCHAR(128) 非 NULL)
主キー列名。

列 5 FKTABLE_CAT (VARCHAR(128))
FKTABLE_NAME のカタログの名前。この表にカタログがない場合、この値は NULL になります。

列 6 FKTABLE_SCHEM (VARCHAR(128))
FKTABLE_NAME を収めたスキーマの名前。

列 7 FKTABLE_NAME (VARCHAR(128) 非 NULL)
外部キーを含む表の名前。

列 8 FKCOLUMN_NAME (VARCHAR(128) 非 NULL)
外部キー列名。

列 9 KEY_SEQ (SMALLINT 非 NULL)
1 から始まるキー内の列の順序を示す位置。

列 10 UPDATE_RULE (SMALLINT)
SQL 操作が UPDATE であるときに外部キーに適用されるアクション。
- SQL_RESTRICT
- SQL_NO_ACTION

IBM DB2 DBMS の更新規則は、常に RESTRICT または SQL_NO_ACTION のいずれかです。しかし、ODBC アプリケーションは IBM 提供でない RDBMS に接続されると、下記の UPDATE_RULE 値を検出する場合があります。
- SQL_CASCADE
- SQL_SET_NULL

列 11 DELETE_RULE (SMALLINT)
SQL 操作が DELETE であるときに外部キーに適用されるアクション。
- SQL_CASCADE
- SQL_NO_ACTION
- SQL_RESTRICT
- SQL_SET_DEFAULT
- SQL_SET_NULL

列 12 FK_NAME (VARCHAR(128))
外部キー ID。データ・ソースに適用できない場合は、NULL。

列 13 PK_NAME (VARCHAR(128))
主キー ID。データ・ソースに適用できない場合は、NULL。

列 14 DEFERRABILITY (SMALLINT)
以下のいずれかです。
- SQL_INITIALLY_DEFERRED
- SQL_INITIALLY_IMMEDIATE
- SQL_NOT_DEFERRABLE

注: CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の `SQLForeignKeys()` 結果セットで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 72. SQLForeignKeys SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 72. SQLForeignKeys SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引数の値が無効です。	引数 *PKTableName* と *FKTableName* はともに NULL ポインターでした。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。 表または所有者名の長さが、サーバーによってサポートされている最大長より長くなっています。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

なし。

例

```
/* get the list of foreign key columns */
cliRC = SQLForeignKeys(hstmt,
                       NULL,
                       0,
                       tbSchema,
                       SQL_NTS,
                       tbName,
                       SQL_NTS,
                       NULL,
                       0,
                       NULL,
                       SQL_NTS,
                       NULL,
                       SQL_NTS);
```

SQLFreeConnect 関数 (CLI) - 接続ハンドルの解放

ODBC 3.0 では SQLFreeConnect() は使用すべきでない関数なので、代わりに SQLFreeHandle() を使用します。

このバージョンの CLI でも引き続き SQLFreeConnect() をサポートしていますが、最新の標準に準拠するように、SQLFreeHandle() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

```
SQLFreeConnect(hdbc);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

SQLFreeEnv 関数 (CLI) - 環境ハンドルの解放

ODBC 3.0 では SQLFreeEnv() は使用すべきでない関数なので、代わりに SQLFreeHandle() を使用します。

このバージョンの CLI でも引き続き SQLFreeEnv() をサポートしていますが、最新の標準に準拠するように、SQLFreeHandle() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

```
SQLFreeEnv(henv);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放

特定の環境、接続、ステートメント、または記述子ハンドルに関連したリソースを解放します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

注: この関数は、リソースを解放するための一般的な関数です。これは、ODBC 2.0 の関数 SQLFreeConnect() (接続ハンドルの解放用) および SQLFreeEnv() (環境ハンドルの解放用) に置き換わる関数です。 SQLFreeHandle() はさらに、ステートメント・ハンドルを解放するための ODBC 2.0 の関数 SQLFreeStmt() (SQL_DROP オプションを使用する) も置き換えます。

構文

```
SQLRETURN   SQLFreeHandle (
               SQLSMALLINT       HandleType,  /* fHandleType */
               SQLHANDLE         Handle);     /* hHandle */
```

関数引数

表 73. SQLFreeHandle 引数

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*HandleType*	入力	SQLFreeHandle() によって解放するハンドルのタイプ。以下の値のうちの 1 つでなければなりません。 • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC *HandleType* が SQL_HANDLE_ENV、SQL_HANDLE_DBC、SQL_HANDLE_STMT、または SQL_HANDLE_DESC 値のいずれでもない場合、SQLFreeHandle() は SQL_INVALID_HANDLE を返します。
SQLHANDLE	*Handle*	入力	解放するハンドル。

使用法

SQLFreeHandle() は、環境、接続、ステートメント、および記述子のハンドルを解放するのに使用されます。

アプリケーションは、ハンドルが解放された後はそのハンドルを使用できません。CLI は、関数呼び出しのハンドルの妥当性チェックは行いません。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLFreeHandle() が SQL_ERROR を返す場合、ハンドルはまだ有効です。

診断

表 74. SQLFreeHandle SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08S01	通信リンクに障害が起きました。	*HandleType* 引数は SQL_HANDLE_DBC であり、 CLI とそれが接続しようとしていたデータ・ソース間の通信リンクは、関数が処理を完了する前に失敗しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。`SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 74. SQLFreeHandle SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	*HandleType* 引数が SQL_HANDLE_ENV であり、少なくとも 1 つの接続が割り当てられているか、接続されている状態にあります。 *HandleType* を SQL_HANDLE_ENV として SQLFreeHandle() を呼び出す前に、 *HandleType* が SQL_HANDLE_DBC である SQLDisconnect() および SQLFreeHandle() を、各接続のために呼び出さなければなりません。 *HandleType* 引数は SQL_HANDLE_DBC であり、関数は、接続のための SQLDisconnect() を呼び出す前に呼び出されました。 *HandleType* 引数は SQL_HANDLE_STMT でした。非同期的に実行する関数がステートメント・ハンドル上で呼び出されました。そして、関数は、この関数が呼び出されたときもまだ実行中でした。 *HandleType* 引数は SQL_HANDLE_STMT でした。 SQLExecute() または SQLExecDirect() はステートメント・ハンドルを使用して呼び出され、 SQL_NEED_DATA が返されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。 (DM) すべての補足的なハンドルと他のリソースは、 SQLFreeHandle() が呼び出される前には解放されませんでした。
HY013	予期しない、メモリーのハンドル・エラーです。	*HandleType* 引数が SQL_HANDLE_STMT または SQL_HANDLE_DESC であり、基礎メモリー・オブジェクトにアクセスできない (メモリー不足が原因と考えられる) ため、関数呼び出しを処理できませんでした。
HY017	自動割り振りの記述子ハンドルについて無効な使用です。	*Handle* 引数が、自動的に割り当てられた記述子またはインプリメンテーション記述子のハンドルに設定されました。

制限

なし。

例

```
/* free the statement handle */
cliRC = SQLFreeHandle(SQL_HANDLE_STMT, hstmt2);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_STMT,
                                       hstmt2,
                                       cliRC,
                                       henv,
                                       hdbc,
                                       pOutSqlrc,
                                       outMsg,
                                       "SQLFreeHandle");
/* ... */
/* free the database handle */
cliRC = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_DBC,
                                       hdbc,
                                       cliRC,
                                       henv,
                                       hdbc,
                                       pOutSqlrc,
                                       outMsg,
                                       "SQLFreeHandle");
```

```
/* free the environment handle */
cliRC = SQLFreeHandle(SQL_HANDLE_ENV, henv);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_ENV,
                                       henv,
                                       cliRC,
                                       henv,
                                       hdbc,
                                       pOutSqlrc,
                                       outMsg,
                                       "SQLFreeHandle");
```

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

ステートメント・ハンドルで参照されているステートメントに関する処理を終了します。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

以下の目的でこの関数を使用します。

- カーソルをクローズし、ペンディング中のすべての結果を破棄します。
- パラメーターをアプリケーション変数と LOB ファイル参照との関連付けから解除 (リセット) します。
- アプリケーション変数からの列と LOB ファイル参照からの列をアンバインドします。
- 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・ハンドルに関連した CLI リソースを解放します。

SQL ステートメントを実行して結果を処理した後、SQLFreeStmt() を呼び出します。

構文

```
SQLRETURN   SQLFreeStmt (SQLHSTMT          StatementHandle,   /* hstmt */
                         SQLUSMALLINT      Option);           /* fOption */
```

関数引数

表 75. SQLFreeStmt 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*Option*	入力	ステートメント・ハンドルの解放の仕方を指定するオプション。このオプションは、以下の値のうちの 1 つでなければなりません。 • SQL_CLOSE • SQL_DROP • SQL_UNBIND • SQL_RESET_PARAMS

使用法

以下のオプションを指定して SQLFreeStmt() を呼び出すことができます。

SQL_CLOSE
ステートメント・ハンドル (*StatementHandle*) に関連したカーソル (存在する場合) がクローズされ、ペンディング状態の結果がすべて廃棄されます。アプリケーションは、*StatementHandle* にバインドされているアプリケーション変数 (存在する場合) と同じ値または別の値を指定して SQLExecute() を呼び出して、カーソルを再オープンします。ステートメント・ハンドルがドロップされるか、または次の SQLGetCursorName() 呼び出しが成功するまで、カーソル名が保持されます。カーソルがステートメント・ハンドルに関連付けされていない場合、このオプションは無効です (警告またはエラーは生成されません)。

SQLCloseCursor() を使用してカーソルをクローズすることもできます。

SQL_DROP
入力ステートメント・ハンドルに関連した CLI リソースが解放され、ハンドルが無効にされます。オープン・カーソル (存在する場合) がクローズされ、ペンディング状態の結果がすべて廃棄されます。

このオプションは、*HandleType* が SQL_HANDLE_STMT に設定された SQLFreeHandle() の呼び出しと置き換えられました。 CLI のこのバージョンでは引き続きこのオプションをサポートしますが、CLI プログラムでは、最新の標準に準拠して SQLFreeHandle() を使用します。

SQL_UNBIND
ARD (アプリケーション行記述子) の SQL_DESC_COUNT フィールドを 0 に設定し、指定されている *StatementHandle* で SQLBindCol() または SQLBindFileToCol() によってバインドされている列バッファーをすべて解放します。これは、ブックマーク列をアンバインドすることはありません。これを行うには、そのブックマーク列の ARD の SQL_DESC_DATA_PTR フィールドを NULL に設定します。この操作が複数のステートメントによって共有されている明示的に割り当てられた記述子で実行される場合、その操作は記述子を共有するステートメントすべてのバインドに影響することに注意してください。

SQL_RESET_PARAMS
APD (アプリケーション・パラメーター記述子) の SQL_DESC_COUNT フィールドを 0 に設定し、指定されている *StatementHandle* で SQLBindParameter() または SQLBindFileToParam() によって設定されているパラメーター・バッファーをすべて解放します。この操作が複数のステートメントによって共有されている明示的に割り当てられた記述子で実行される場合、その操作は記述子を共有するステートメントすべてのバインドに影響することに注意してください。

SQLFreeStmt() は LOB ロケーターには無効で、 FREE LOCATOR ステートメントを指定して SQLExecDirect() を呼び出し、ロケーターを解放します。

以下のように、ステートメント・ハンドルを再使用して別のステートメントを実行することができます。

- ハンドルが照会、カタログ関数、または SQLGetTypeInfo() に関連付けられていた場合、カーソルをクローズする必要があります。
- ハンドルが異なる数またはタイプのパラメーターにバインドされていた場合は、パラメーターをリセットしなければなりません。
- ハンドルが、異なる数またはタイプの列バインディングにバインドされていた場合は、列をアンバインドしなければなりません。

別の方法として、ステートメント・ハンドルをドロップし、新しいステートメント・ハンドルを割り振ることができます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Option を SQL_DROP に設定すると SQL_SUCCESS_WITH_INFO は返されませんが、それは、 SQLGetDiagRec() または SQLGetDiagField() の呼び出し時には使用するステートメント・ハンドルがないからです。

診断

表 76. SQLFreeStmt SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。
HY092	オプション・タイプが範囲外です。	引数 *Option* に指定された値は、SQL_CLOSE、SQL_DROP、SQL_UNBIND、または SQL_RESET_PARAMS のどれでもありませんでした。
HY506	ファイルのクローズ・エラーです。	一時ファイルをクローズしようとしているときにエラーが発生しました。

許可

なし。

例

```
/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_UNBIND);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
```

```
  return 1;
}

/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_RESET_PARAMS);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
  return 1;
}

/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_CLOSE);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
  return 1;
}
```

SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得

接続属性の現在の設定を返します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetConnectAttrW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN   SQLGetConnectAttr(SQLHDBC           ConnectionHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        BufferLength,
                              SQLINTEGER        *StringLengthPtr);
```

関数引数

表 77. SQLGetConnectAttr 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLINTEGER	*Attribute*	入力	取り出す属性。
SQLPOINTER	*ValuePtr*	出力	*属性* によって指定されている属性の現行値を返すメモリーを指すポインター。

表 77. SQLGetConnectAttr 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER	*BufferLength*	入力	• *ValuePtr* が文字ストリングを指す場合、この引数の長さは **ValuePtr* になります。 • *ValuePtr* がポインターであってもストリングを指していない場合、 *BufferLength* の値は SQL_IS_POINTER になります。 • **ValuePtr* の値が Unicode ストリングである場合、*BufferLength* 引数は偶数でなければなりません。
SQLINTEGER *	*StringLengthPtr*	出力	**ValuePtr* 内に戻すために使用できる総バイト数 (NULL 終止符文字を除く) を戻すバッファーを指すポインター。 *ValuePtr* が NULL ポインターである場合、長さは戻されません。属性値が文字ストリングであり、返すことが可能なバイトの数が BufferLength から NULL 終止符文字の長さを引いたものより大きい場合、 **ValuePtr* のデータは、 *BufferLength* から NULL 終止符文字分を減算した長さに切り捨てられ、 CLI によってヌル終了になります。

使用法

属性 がストリングを返す属性を指定する場合、*ValuePtr* は、そのストリングのバッファーを指すポインターでなければなりません。 NULL 終止符文字を含めたストリングの最大長は、*BufferLength* バイトになります。

属性によっては、SQLGetConnectAttr() を呼び出す前に接続を構築する必要のないアプリケーションがあります。しかし、SQLGetConnectAttr() が呼び出され、指定した属性にデフォルト値がなく、 SQLSetConnectAttr() より前の呼び出しによって設定されていない場合は、 SQLGetConnectAttr() は SQL_NO_DATA を返します。

属性 が SQL_ATTR_TRACE であるか、または SQL_ATTR_TRACEFILE である場合、 *ConnectionHandle* は有効である必要はなく、 *ConnectionHandle* が無効である場合、 SQLGetConnectAttr() は SQL_ERROR を返すことはありません。これらの属性は、すべての接続に適用されます。別の引数が無効である場合、SQLGetConnectAttr() は SQL_ERROR を返します。

アプリケーションが SQLSetConnectAttr() を使用してステートメント属性を設定できるときに、アプリケーションはステートメント属性値を取り出すために SQLGetConnectAttr() を使用することはできません。ステートメント属性の設定を取り出すには、SQLGetStmtAttr() を呼び出さなければなりません。

SQL_ATTR_AUTO_IPD 接続属性は、SQLGetConnectAttr() の呼び出しによって返されますが、 SQLSetConnectAttr() の呼び出しによって設定することはできません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 78. SQLGetConnectAttr SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	**ValuePtr* に戻されるデータは、 *BufferLength* から NULL 終止符文字の長さを引いた長さに切り捨てられます。 **StringLengthPtr* には、切り捨て前のストリング値の長さが戻されます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08003	接続がクローズされています。	オープン接続が必要な *Attribute* 値を指定しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	*ConnectionHandle* で SQLBrowseConnect() が呼び出され、SQL_NEED_DATA が戻されました。この関数は、SQLBrowseConnect() が SQL_SUCCESS_WITH_INFO または SQL_SUCCESS を戻す前に呼び出されました。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* に指定された値は、0 より小さい値でした。
HY092	オプション・タイプが範囲外です。	引数 *Attribute* に指定された値は、無効でした。
HYC00	ドライバーが使用できません。	引数 *Attribute* に指定された値は、このバージョンの CLI ドライバーには有効な接続またはステートメント属性でしたが、データ・ソースではサポートされていませんでした。

制限

なし。

例

```
SQLINTEGER autocommit;

/* ... */

/* get the current setting for the AUTOCOMMIT attribute */
cliRC = SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, &autocommit, 0, NULL);
```

SQLGetConnectOption 関数 (CLI) - 接続オプションの現行設定値を戻す

ODBC バージョン 3 では SQLGetConnectOption() は使用すべきでないので、代わりに SQlGetConnectAttr() を使用します。

このバージョンの CLI でも引き続き SQLGetConnectOption() をサポートしていますが、最新の標準に準拠するように、SQLGetConnectAttr() を CLI プログラムで使用します。

新しい関数へのマイグレーション

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetConnectOptionW() です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

例えば、次のようなステートメントを想定します。

```
SQLGetConnectOption(hdbc, SQL_ATTR_AUTOCOMMIT, pvAutoCommit);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, pvAutoCommit,
                  SQL_IS_POINTER, NULL);
```

SQLGetCursorName 関数 (CLI) - カーソル名の取得

入力ステートメント・ハンドルに関連したカーソル名を返します。

SQLSetCursorName() を呼び出してカーソル名を明示設定すると、このカーソル名が返されます。それ以外の場合は暗黙生成された名前が返されます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetCursorNameW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN   SQLGetCursorName (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLCHAR           *CursorName,      /* szCursor */
               SQLSMALLINT       BufferLength,     /* cbCursorMax */
               SQLSMALLINT       *NameLengthPtr);  /* pcbCursor */
```

関数引数

表 79. SQLGetCursorName 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CursorName*	出力	カーソル名
SQLSMALLINT	*BufferLength*	入力	*CursorName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。

表 79. SQLGetCursorName 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*NameLengthPtr*	出力	*CursorName* に戻すために使用できる SQLCHAR エレメント (この関数の Unicode 版の場合は SQLWCHAR エレメント) の数 (NULL 終止符文字を除く)。

使用法

SQLGetCursorName() は、 SQLSetCursorName() で明示的に設定されたカーソル名を返すか、または名前が設定されていない場合は、CLI によって内部で生成されたカーソル名を返します。入力ステートメント・ハンドル上でステートメント・ハンドルの準備が完了する前に SQLGetCursorName() を呼び出すと、エラーが生じます。ステートメント・ハンドル上で内部カーソル名が生成されるのは、そのハンドルの割り振り時ではなく、ステートメント・ハンドル上で最初に動的 SQL が準備されるときです。

SQLSetCursorName() を使用して名前を明示設定すると、ステートメントをドロップするか、別の明示的な名前を設定するまで、この名前が返されます。

内部で生成されたカーソル名は、常に SQLCUR または SQL_CUR で始まります。カーソル名の SQLCHAR 数または SQLWCHAR 数は常に 128 以下であり、接続中は常にユニークです。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 80. SQLGetCursorName SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	*CursorName* 内に返されたカーソル名は *BufferLength* 内の値より長かったので、 *BufferLength* - 1 バイトに切り捨てられます。引数 *NameLengthPtr* には、戻りに使用できる完全カーソル名の長さが含まれています。関数は、 SQL_SUCCESS_WITH_INFO を返します。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。

表 80. SQLGetCursorName SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* に指定された値は 0 より小さい値です。

制限

ODBC 生成のカーソル名は SQL_CUR で始まり、CLI 生成のカーソル名は SQLCUR で始まり、X/Open CLI 生成のカーソル名は SQLCUR または SQL_CUR で始まります。

例

```
SQLCHAR cursorName[20];

/* ... */

/* get the cursor name of the SELECT statement */
cliRC = SQLGetCursorName(hstmtSelect, cursorName, 20, &cursorLen);
```

SQLGetData 関数 (CLI) - 列からのデータの取得

結果セットの現在行で 1 つの列のデータを取り出します。

この関数は SQLBindCol() の代わりになるものであり、 SQLFetch() または SQLFetchScroll() 呼び出しでアプリケーション変数または LOB ロケーターへデータを直接転送するのに使用されます。アプリケーションは、LOB を SQLBindCol() でバインド、または SQLGetData() を使用して LOB を検索できますが、両方の方式を同時に使用することはできません。 SQLGetData() を使用して、大きなデータ値を分割して取り出すこともできます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

SQLFetch() または SQLFetchScroll() は、SQLGetData() の前に呼び出す必要があります。

列ごとに SQLGetData() を呼び出した後で、 SQLFetch() または SQLFetchScroll() を呼び出して以下の行を取り出します。

構文

```
SQLRETURN   SQLGetData      (
               SQLHSTMT         StatementHandle,     /* hstmt */
               SQLUSMALLINT     ColumnNumber,        /* icol */
               SQLSMALLINT      TargetType,          /* fCType */
               SQLPOINTER       TargetValuePtr,      /* rgbValue */
               SQLLEN           BufferLength,        /* cbValueMax */
               SQLLEN           *StrLen_or_IndPtr);  /* pcbValue */
```

関数引数

表 81. SQLGetData 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*ColumnNumber*	入力	データ取り出しが要求されている列番号。結果セット列は、左から右へ順番に番号が付けられています。 • ブックマークを使用していない場合 (SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF)、列番号は 1 から始まります。 • ブックマークを使用している場合 (そのステートメント属性が SQL_UB_ON または SQL_UB_VARIABLE)、列番号は 0 から始まります。

表 81. SQLGetData 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*TargetType*	入力	*ColumnNumber* による列 ID の C データ・タイプ。以下のタイプがサポートされます。 • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DECIMAL_IBM • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_NUMERIC [a] • SQL_C_SBIGINT • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TYPE_TIMESTAMP_EXT • SQL_C_TINYINT • SQL_C_UBIGINT • SQL_C_UTINYINT • SQL_C_WCHAR SQL_ARD_TYPE を指定すると、データは、 ARD の SQL_DESC_CONCISE_TYPE フィールドに指定されているデータ・タイプに変換されることになります。 SQL_C_DEFAULT を指定するとデータは、デフォルトの C データ・タイプに変換されます。
SQLPOINTER	*TargetValuePtr*	出力	検索された列データを格納するバッファーを指すポインター。
SQLLEN	*BufferLength*	入力	*TargetValuePtr* で指し示されたバッファーの最大サイズ。ドライバーが固定長データを戻すと、この値は無視されます。

表 81. SQLGetData 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLLEN *	*StrLen_or_IndPtr*	出力	CLI が *TargetValuePtr* バッファー内に返すのに使用できるバイト数を示す値を指すポインター。データが分割して検索されている場合、これにはまだ残っているバイト数が入ります。 列のデータ値が NULL の場合、値は SQL_NULL_DATA です。このポインターが NULL で、 SQLFetch() で NULL データを含む列を取得した場合、この関数はこのことを報告する手段がないので失敗します。 SQLFetch() がバイナリー・データを含む列を取り出した場合、 *StrLen_or_IndPtr* を指すポインターが NULL であってはなりません。 NULL の場合、この関数は *TargetValuePtr* バッファーに取り出されたデータの長さについてアプリケーションに通知する他の手段がないので失敗します。

注: *TargetValuePtr* がメモリー内の *StrLen_or_IndPtr* の後に連続して設定されると、CLI のパフォーマンスは多少強化されます。

使用法

DB2 データ・ソースが異なれば、SQLGetData() の使用法に関する制限も異なります。アプリケーションは、この関数の機能を確認するには、以下の SQL_GETDATA_EXTENSIONS オプションのいずれかを指定して SQLGetInfo() を呼び出す必要があります。

- SQL_GD_ANY_COLUMN: このオプションが戻された場合、最後にバインドされた列より前のものも含め、すべてのアンバインド済み列に対して SQLGetData() を呼び出すことができます。すべての DB2 データ・ソースがこのフィーチャーをサポートします。
- SQL_GD_ANY_ORDER: このオプションが戻された場合、任意の順序でアンバインド済み列に対して SQLGetData() を呼び出すことができます。すべての DB2 データ・ソースがこのフィーチャーをサポートします。
- SQL_GD_BLOCK: SQL_GETDATA_EXTENSIONS *InfoType* 引数の場合に SQLGetInfo() でこのオプションが戻された場合、行セットのサイズが 1 より大きければ、ドライバーは SQLGetData() の呼び出しをサポートします。アプリケーションは SQL_POSITION オプションを指定した SQLSetPos() を呼び出して、カーソルを正しい行上に置いてから SQLGetData() を呼び出すこともできます。少なくとも DB2 (UNIX および Windows 版) のデータ・ソースは、このフィーチャーをサポートします。
- SQL_GD_BOUND: このオプションが戻された場合、バインド済み列ならびにアンバインド済み列に対して SQLGetData() を呼び出すことができます。 DB2 Database for Linux, UNIX, and Windows は現在このフィーチャーをサポートしていません。

C データ・タイプ (*TargetType*) が SQL_C_CHAR、 SQL_C_BINARY、 SQL_C_DBCHAR、 SQL_C_WCHAR であるか、または *TargetType* が

SQL_C_DEFAULT で列タイプがバイナリーまたは文字ストリングを示している場合、 SQLGetData() を使用して長い列を取り出すこともできます。

SQLGetData() の呼び出しごとに、戻りに使用できるデータが *BufferLength* 以上の場合には、切り捨てが行われます。切り捨ては、関数戻りコード SQL_SUCCESS_WITH_INFO とデータ切り捨てを示す SQLSTATE で示されます。アプリケーションは、同じ ColumnNumber 値を指定して SQLGetData() を再度呼び出し、アンバインドされた同じ列から切り捨て時以降のデータを取得することができます。列全体を取得するには、関数が SQL_SUCCESS を返すまでアプリケーションがこのような呼び出しを繰り返します。次の SQLGetData() 呼び出しは、SQL_NO_DATA_FOUND を返します。

アプリケーションが関数 SQLGetData() を呼び出して実際の LOB データを取り出す場合、デフォルトで、サーバーに 1 つの要求を行い、*BufferLength* の大きさが十分である場合に、LOB 全体をメモリーに保管します。*BufferLength* が、要求された LOB データを保留できるほど大きくない場合には、分割して取り出されます。

SQLGetData() は LOB 列データの順次検索に使用できますが、 LOB データのごく一部または LOB 列データの少数のセクションが必要な場合には、 CLI LOB 関数を使用してください。

1. LOB ロケーターに列をバインドします。
2. 行をフェッチします。
3. SQLGetSubString() 呼び出しにロケーターを使用して、データを分割して検索してください (一部の引数の値を判別するために、 SQLGetLength() および SQLGetPosition() が必要になることがあります)。
4. ステップ 2 を繰り返します。

切り捨ては、SQL_ATTR_MAX_LENGTH ステートメント属性によっても影響されます。アプリケーションは、 SQL_ATTR_MAX_LENGTH および列ごとに返される最大長の値を指定して SQLSetStmtAttr() を呼び出し、同サイズ (に NULL 終止符文字分を加えたもの) の *TargetValuePtr* バッファーを割り振って、切り捨てが報告されないように指定することができます。列データが設定された最大長より大きい場合、SQL_SUCCESS が返され、実際の長さではなく最大長が *StrLen_or_IndPtr* に返されます。

取り出しによって列データの部分を廃棄するために、アプリケーションは *ColumnNumber* を次の対象列の位置に設定して SQLGetData() を呼び出すことができます。行全体として取り出されなかったデータを廃棄するには、アプリケーションで SQLFetch() を呼び出してカーソルを次の行に進めなければなりませんが、結果セットからのデータがこれ以上必要ない場合は、 SQL_CLOSE または SQL_DROP を指定した `SQLCloseCursor()` または `SQLFreeStmt()` を呼び出してカーソルをクローズできます。

TargetType 入力引数は、 *TargetValuePtr* で指示されたストレージに列データを入れる前に、必要なデータ変換 (存在する場合) のタイプを判別します。

SQL GRAPHIC 列データの場合、以下のようになります。

- *TargetValuePtr* バッファーの長さ (*BufferLength*) は、2 の倍数にします。アプリケーションは、最初に SQLDescribeCol() または SQLColAttribute() を呼び出して、列の SQL データ・タイプを判別することができます。
- CLI は *StrLen_or_IndPtr* 内に保管されているオクテット数を保管するので、*TargetValuePtr* を指すポインターを NULL にすることはできません。
- データを分割して取り出す場合、 CLI は *TargetValuePtr* の値以下の最も大きい 2 オクテットの倍数まで *BufferLength* を埋め込もうとします。このことは、*BufferLength* が 2 の倍数でない場合にそのバッファー内の最後のバイトには処理を行わないことを意味します。 CLI は、2 バイト文字を分割しません。

検索する列データがバイナリーでないか、または列の SQL データ・タイプが GRAPHIC (DBCS) であって C バッファー・タイプが SQL_C_CHAR であると、*TargetValuePtr* に返される内容は常にヌル終了です。アプリケーションが複数の部分に分けてデータを検索している場合は、適切な調整を加える必要があります (例えば、ヌル終了環境属性が有効であると想定して、各部分を連結し直す前に NULL 終止符文字を除去します)。

切り捨てが小数点の右側の桁数に関係している場合、数値データ・タイプの切り捨ては警告として報告されます。切り捨てが小数点の左側で行われると、エラーが返されます (診断のセクションを参照)。

両方向スクロール・カーソルは例外ですが、データを検索するのに SQLFetchScroll() を使用するアプリケーションが SQLGetData() を呼び出すべきなのは、行セット・サイズが 1 (SQLFetch() を発行するのと同じ) のときだけです。 SQLGetData() は、カーソルが現在置かれている行の列データだけを取り出せます。

両方向スクロール・カーソルでの SQLGetData() の使用

SQLGetData() は両方向スクロール・カーソルとともに使用することもできます。結果セットにある任意の行へのポインターを、ブックマークを付けて保管することができます。アプリケーションは、そのブックマークを相対位置として使用して、情報の行セットを検索します。

SQLSetPos() を使用して、行セット内の行へカーソルをいったん位置決めすれば、SQLGetData() を使用して列 0 からブックマーク値を得ることができます。多くの場合、列 0 をバインドして行ごとのブックマーク値を検索する必要はありませんが、 SQLGetData() を使用すると必要な特定行のブックマーク値を検索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
- SQL_NO_DATA_FOUND
- SQL_NO_TOTAL

先行の SQLGetData() 呼び出しでこの列のすべてのデータが検索されると、SQL_NO_DATA_FOUND が返されます。

SQLGetData() で長さゼロのストリングが取り出されると、SQL_SUCCESS が返されます。この場合、StrLen_or_IndPtr に 0 が入れられ、TargetValuePtr に NULL 終止符文字が入れられます。

CLI 構成キーワード StreamGetData が 1 に設定されており、CLI が出力バッファー戻すためにまだ使用できるバイト数を判別できない場合、切り捨て発生時に長さとして SQL_NO_TOTAL が戻されます。

直前の SQLFetch() 呼び出しが失敗した場合、結果は定義されないので SQLGetData() を呼び出すことはできません。

診断

表 82. SQLGetData SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	指定された列 (ColumnNumber) に返されたデータは切り捨てられました。ストリングまたは数値は右方切り捨てされます。SQL_SUCCESS_WITH_INFO が返されます。
07006	無効な変換です。	引数 *TargetType* で指定された C データ・タイプにデータ値を変換することはできません。 この関数は以前に同じ *ColumnNumber* 値に対して呼び出されましたが、 *TargetType* 値が異なっています。
07009	記述子索引が無効です。	*ColumnNumber* に指定された値が 0 と同等であり、SQL_ATTR_USE_BOOKMARKS ステートメント属性が SQL_UB_OFF でした。引数 *ColumnNumber* に指定された値は、結果セット内の列数より大きい値でした。
22002	無効な出力または標識バッファーが指定されました。	引数 StrLen_or_IndPtr に指定されたポインター値は NULL ポインターで、列の値は NULL です。 SQL_NULL_DATA を報告する手段はありません。
22003	数値が範囲外です。	列の数値を (数値またはストリングとして) 返したため、数値の整数部分が切り捨てられたと考えられます。
22005	割り当てにエラーがありました。	戻り値は、引数 *TargetType* で指示されたデータ・タイプと互換性がありませんでした。
22007	無効な日付時刻形式です。	文字ストリングから日時フォーマットへの変換が指定されましたが、無効なストリング表示または値が指定されたか、あるいは値が無効な日付になっています。
22008	日時フィールドがオーバーフローしました。	日時フィールドのオーバーフローが発生しました。例えば、日付またはタイム・スタンプの算術計算の結果が有効な日付範囲内の値にならないか、またはバインドされた変数が小さすぎるため日時値を代入できません。
24000	カーソル状態が無効です。	直前の SQLFetch() の結果が SQL_ERROR であったか、または SQL_NO_DATA が検出されました。その結果、カーソルは行に置かれていません。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。

表 82. SQLGetData SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY003	プログラム・タイプが範囲外です。	*TargetType* は、有効なデータ・タイプまたは SQL_C_DEFAULT ではありませんでした。
HY010	関数のシーケンス・エラーです。	指定された *StatementHandle* がカーソル定位置状態にありませんでした。最初に SQLFetch を呼び出さずに、この関数を呼び出しました。 実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY011	この時点で無効な操作です。	以前にアクセスした LOB 列について SQLGetData() を呼び出すことは許可されていません。詳しくは、 364 ページの『AllowGetDataLOBReaccess CLI/ODBC 構成キーワード』 を参照してください。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* の値は 0 より小さく、引数 *TargetType* は SQL_C_CHAR、SQL_C_BINARY、 SQL_C_DBCHAR (または SQL_C_DEFAULT で、デフォルト・タイプ SQL_C_CHAR、SQL_C_BINARY、または SQL_C_DBCHAR のいずれか) です。
HYC00	ドライバーが使用できません。	指定されたデータ・タイプの SQL データ・タイプは認識されますが、CLI ではサポートされません。 SQL データ・タイプからアプリケーション・データ *TargetType* への要求された変換を、CLI またはデータ・ソースで行うことはできません。 列は SQLBindFileToCol() を使用してバインドされました。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

例

```
/* use SQLGetData to get the results */
/* get data from column 1 */
cliRC = SQLGetData(hstmt,
                   1,
                   SQL_C_SHORT,
                   &deptnumb.val,
                   0,
                   &deptnumb.ind);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* get data from column 2 */
cliRC = SQLGetData(hstmt,
                   2,
                   SQL_C_CHAR,
                   location.val,
                   15,
                   &location.ind);
```

SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得

記述子レコードの単一のフィールドの現行設定を返します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetDescFieldW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLGetDescField (
               SQLHDESC          DescriptorHandle,
               SQLSMALLINT       RecNumber,
               SQLSMALLINT       FieldIdentifier,
               SQLPOINTER        ValuePtr,           /* Value */
               SQLINTEGER        BufferLength,
               SQLINTEGER        *StringLengthPtr);  /* *StringLength */
```

関数引数

表 83. SQLGetDescField 引数

データ・タイプ	引数	使用法	説明
SQLHDESC	*DescriptorHandle*	入力	記述子ハンドル。

表 83. SQLGetDescField 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*RecNumber*	入力	アプリケーションによって検索される情報を収めた記述子レコードを指定します。記述子レコードは 0 から数え、レコード番号 0 がブックマーク・レコードになります。 *FieldIdentifier* 引数が記述子ヘッダー・レコードのフィールドを指定する場合、*RecNumber* は 0 でなければなりません。*RecNumber* が SQL_DESC_COUNT より小さく、行に列またはパラメーターのデータが含まれない場合、SQLGetDescField() の呼び出しはフィールドのデフォルト値を返します。
SQLSMALLINT	*FieldIdentifier*	入力	値を返す記述子のフィールドを指定します。
SQLPOINTER	*ValuePtr*	出力	記述子情報を返す先のバッファーを指すポインターです。データ・タイプは、*FieldIdentifier* の値により異なります。
SQLINTEGER	*BufferLength*	入力	• *ValuePtr* が文字ストリングを指す場合、この引数の長さは **ValuePtr* になります。 • *ValuePtr* がポインターであってもストリングを指していない場合、 *BufferLength* の値は SQL_IS_POINTER になります。 • **ValuePtr* の値が Unicode データ・タイプである場合、 *BufferLength* 引数は偶数でなければなりません。
SQLSMALLINT *	*StringLengthPtr*	出力	**ValuePtr* 内に戻すために使用できる総バイト数 (NULL 終止符文字に必要なバイト数を除く) を指すポインター。

使用法

アプリケーションは、SQLGetDescField() を呼び出して、記述子レコードの単一フィールドの値を返すことができます。 SQLGetDescField() への呼び出しでは、ヘッダー・フィールド、レコード・フィールド、およびブックマーク・フィールドを含むすべての記述子タイプのどのフィールドの設定でも返すことができます。SQLGetDescField() を繰り返し呼び出すなら、アプリケーションは同じ記述子または異なる記述子にある複数のフィールドの設定を、任意の順序で獲得することができます。 SQLGetDescField() を呼び出して CLI 定義済み記述子フィールドを返すこともできます。

パフォーマンス上の理由で、ステートメントを実行する前にアプリケーションは IRD の SQLGetDescField() を呼び出してはなりません。そのような時点で SQLGetDescField() を呼び出すと、CLI ドライバーがステートメントを記述するので、さらに余計なネットワーク・フローを生じることになります。据え置き準備がオンになっているときに SQLGetDescField() を呼び出すと、記述情報を得るためにサーバーでステートメントを準備する必要が生じるため、据え置き準備の利点が失われます。

名前、データ・タイプ、および列またはパラメーター・データのストレージを記述できる複数のフィールドの設定を、 SQLGetDescRec() への単一呼び出しで検索する

こともできます。 SQLGetStmtAttr() を呼び出して、ステートメント属性が関連付けられた記述子ヘッダー内の単一のフィールドの値を返すことができます。

特定の記述子タイプが未定義になっているフィールドの値を検索するためにアプリケーションが SQLGetDescField() を呼び出すと、この関数は SQLSTATE HY091 (無効な記述子フィールド ID) を返します。特定の記述子タイプが定義されているのに、デフォルト値をもっておらず、しかもまだ設定されていないフィールドの値を検索するためにアプリケーションが SQLGetDescField() を呼び出すと、この関数は SQL_SUCCESS を返しますがフィールドに返される値は未定義になります。存在するデフォルト値を確かめるには、記述子フィールドの初期化値のリストを参照してください。

SQL_DESC_ALLOC_TYPE ヘッダー・フィールドは、読み取り専用として使用できます。このフィールドは、すべてのタイプの記述子に定義されます。

これらのフィールドはそれぞれ、IRD 専用か、または IRD と IPD の両方のどちらかに定義されます。

```
SQL_DESC_AUTO_UNIQUE_VALUE    SQL_DESC_LITERAL_SUFFIX
SQL_DESC_BASE_COLUMN_NAME     SQL_DESC_LOCAL_TYPE_NAME
SQL_DESC_CASE_SENSITIVE       SQL_DESC_SCHEMA_NAME
SQL_DESC_CATALOG_NAME         SQL_DESC_SEARCHABLE
SQL_DESC_DISPLAY_SIZE         SQL_DESC_TABLE_NAME
SQL_DESC_FIXED_PREC_SCALE     SQL_DESC_TYPE_NAME
SQL_DESC_LABEL                SQL_DESC_UNSIGNED
SQL_DESC_LITERAL_PREFIX       SQL_DESC_UPDATABLE
```

上記のフィールドの詳細は、記述子 *FieldIdentifier* の値のリストを参照してください。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_NO_DATA
- SQL_INVALID_HANDLE

RecNumber が記述子レコードの数よりも大きい場合は、SQL_NO_DATA が返されます。

DescriptorHandle が IRD ハンドルであり、ステートメントが準備済みまたは実行済み状態にあるが、それと関連するオープン・カーソルがない場合、SQL_NO_DATA が返されます。

診断

表 84. SQLGetDescField SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	バッファー **ValuePtr* には記述子フィールド全体を返すのに十分な大きさがなかったため、フィールドが切り捨てられました。切り捨てられていない記述子フィールドの長さが、**StringLengthPtr* に返されます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)

表 84. SQLGetDescField SQLSTATE (続き)

SQLSTATE	説明	解説
07009	記述子索引が無効です。	*RecNumber* 引数に指定された値は 1 より小さく、SQL_ATTR_USE_BOOKMARKS ステートメント属性は SQL_UB_OFF であり、フィールドはヘッダー・フィールドまたは CLI 定義済みフィールドではありませんでした。 *FieldIdentifier* 引数はレコード・フィールドであり、 *RecNumber* 引数は 0 でした。 *RecNumber* 引数は 0 より小さく、フィールドはヘッダー・フィールドまたは CLI 定義済みフィールドではありませんでした。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY007	関連ステートメントが準備されていません。	*DescriptorHandle* は IRD と関連付けられており、関連付けられているステートメント・ハンドルが準備済みまたは実行済みの状態にはありません。
HY010	関数のシーケンス・エラーです。	*DescriptorHandle* が関連付けられていた StatementHandle で、非同期的に実行されるある関数 (この関数ではない) が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *DescriptorHandle* に関連する *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が返されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY021	記述子情報が矛盾します。	整合性チェック時にチェックされた記述子情報は、整合性がとれていませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HY091	記述子フィールド ID が無効です。	*DescriptorHandle* には *FieldIdentifier* が定義されていません。 SQL_DESC_COUNT フィールド内の値よりも大きい値が *RecNumber* 引数に指定されました。

制限

なし。

例

```
/* see how the field SQL_DESC_PARAMETER_TYPE is set */
cliRC = SQLGetDescField(hIPD,
                        1, /* look at the parameter */
                        SQL_DESC_PARAMETER_TYPE,
                        &descFieldParameterType, /* result */
                        SQL_IS_SMALLINT,
                        NULL);

/* ... */

  /* see how the descriptor record field SQL_DESC_TYPE_NAME is set */
  rc = SQLGetDescField(hIRD,
                       (SQLSMALLINT)colCount,
                       SQL_DESC_TYPE_NAME, /* record field */
                       descFieldTypeName, /* result */
                       25,
                       NULL);
  /* ... */

  /* see how the descriptor record field SQL_DESC_LABEL is set */
  rc = SQLGetDescField(hIRD,
                       (SQLSMALLINT)colCount,
                       SQL_DESC_LABEL, /* record field */
                       descFieldLabel, /* result */
                       25,
                       NULL);
```

SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得

記述子レコードの複数フィールドの現行設定を返します。

返されたフィールドは、名前、データ・タイプ、および列またはパラメーター・データのストレージを記述します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetDescRecW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLGetDescRec (
               SQLHDESC          DescriptorHandle,   /* hDesc */
               SQLSMALLINT       RecNumber,
               SQLCHAR           *Name,
               SQLSMALLINT       BufferLength,
               SQLSMALLINT       *StringLengthPtr,   /* *StringLength */
               SQLSMALLINT       *TypePtr,           /* *Type */
               SQLSMALLINT       *SubTypePtr,        /* *SubType */
               SQLLEN            *LengthPtr,         /* *Length */
```

```
SQLSMALLINT       *PrecisionPtr,      /* *Precision */
SQLSMALLINT       *ScalePtr,          /* *Scale */
SQLSMALLINT       *NullablePtr);      /* *Nullable */
```

関数引数

表 85. SQLGetDescRec 引数

データ・タイプ	引数	使用法	説明
SQLHDESC	*DescriptorHandle*	入力	記述子ハンドル。
SQLSMALLINT	*RecNumber*	入力	アプリケーションによって検索される情報を収めた記述子レコードを指定します。記述子レコードは 0 から数え、レコード番号 0 がブックマーク・レコードになります。 *RecNumber* 引数は、SQL_DESC_COUNT の値以下でなければなりません。 *RecNumber* が SQL_DESC_COUNT より小さくても、行に列またはパラメーターのデータが入っていないと、 SQLGetDescRec() の呼び出しはフィールドのデフォルト値を返します。
SQLCHAR *	*Name*	出力	記述子レコードの SQL_DESC_NAME フィールドを返す先のバッファーを指すポインターです。
SQLINTEGER	*BufferLength*	入力	**Name* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLSMALLINT *	*StringLengthPtr*	出力	*Name* に戻すために使用できる SQLCHAR エレメントの数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの数) を戻すバッファーを指すポインター (NULL 終止符文字を除く)。SQLCHAR または SQLWCHAR エレメントの数が *BufferLength* 以上の場合、 **Name* のデータは、*BufferLength* から NULL 終止符文字の長さを減算した長さに切り捨てられ、 CLI によってヌル終了ストリングになります。
SQLSMALLINT *	*TypePtr*	出力	記述子レコードの SQL_DESC_TYPE フィールドの値を返す先のバッファーを指すポインターです。
SQLSMALLINT *	*SubTypePtr*	出力	SQL_DATETIME のタイプをもつレコードの場合に、SQL_DESC_DATETIME_INTERVAL_CODE フィールドの値を返す先のバッファーを指すポインターです。
SQLLEN *	*LengthPtr*	出力	記述子レコードの SQL_DESC_OCTET_LENGTH フィールドの値を返す先のバッファーを指すポインターです。
SQLSMALLINT *	*PrecisionPtr*	出力	記述子レコードの SQL_DESC_PRECISION フィールドの値を返す先のバッファーを指すポインターです。
SQLSMALLINT *	*ScalePtr*	出力	記述子レコードの SQL_DESC_SCALE フィールドの値を返す先のバッファーを指すポインターです。
SQLSMALLINT *	*NullablePtr*	出力	記述子レコードの SQL_DESC_NULLABLE フィールドの値を返す先のバッファーを指すポインターです。

使用法

アプリケーションは、SQLGetDescRec() を呼び出して、以下のフィールドにおいて単一の列またはパラメーターの値を検索することができます。

- SQL_DESC_NAME
- SQL_DESC_TYPE
- SQL_DESC_DATETIME_INTERVAL_CODE (タイプが SQL_DATETIME のレコード)
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_NULLABLE

SQLGetDescRec() は、ヘッダー・フィールドの値は検索しません。

アプリケーションは、 NULL ポインターのフィールドに対応する引数を設定することにより、フィールドの設定を返すことを禁止することができます。特定の記述子タイプが未定義になっているフィールドの値を検索するためにアプリケーションが SQLGetdescRec() を呼び出すとき、この関数は SQL_SUCCESS を返しますが、フィールドに対して返される値は定義されていません。例えば、APD または ARD の SQL_DESC_NAME または SQL_DESC_NULLABLE に対して SQLGetDescRec() を呼び出すと SQL_SUCCESS が返されますが、フィールドには未定義の値が返されます。

特定の記述子タイプ用に定義済みフィールドの値を検索するためにアプリケーションが SQLGetDescRec() を呼び出したものの、デフォルト値がなく、まだ設定されていないときには、この関数は SQL_SUCCESS を返しますが、フィールドに対して返される値は定義されていません。

フィールドの値は、SQLGetDescField() の呼び出しによって個々に検索することもできます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_NO_DATA
- SQL_INVALID_HANDLE

RecNumber が記述子レコードの数よりも大きい場合は、SQL_NO_DATA が返されます。

DescriptorHandle が IRD ハンドルであり、ステートメントが準備済みまたは実行済み状態にあるが、それと関連するオープン・カーソルがない場合、SQL_NO_DATA が返されます。

診断

表 86. SQLGetDescRec SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	バッファー **Name* には記述子フィールド全体を返すのに十分な大きさがなかったため、フィールドが切り捨てられました。切り捨てられていない記述子フィールドの長さが、**StringLengthPtr* に返されます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
07009	記述子索引が無効です。	*RecNumber* 引数は 0 に設定され、 *DescriptorHandle* 引数は IPD ハンドルでした。 *RecNumber* 引数は 0 に設定され、SQL_ATTR_USE_BOOKMARKS ステートメント属性は SQL_UB_OFF に設定されました。 *RecNumber* 引数は 0 未満でした。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY007	関連ステートメントが準備されていません。	*DescriptorHandle* は IRD と関連付けられており、関連付けられているステートメント・ハンドルが準備済みまたは実行済みの状態にはありません。
HY010	関数のシーケンス・エラーです。	*DescriptorHandle* が関連付けられていた *StatementHandle* で、非同期的に実行されるある関数 (この関数ではない) が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *DescriptorHandle* に関連する *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が返されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。

制限

なし。

例

```
/* get multiple field settings of descriptor record */
rc = SQLGetDescRec(hIRD,
                   i,
                   colname,
                   sizeof(colname),
```

```
                    &namelen,
                    &type,
                    &subtype,
                    &width,
                    &precision,
                    &scale,
                    &nullable);

/* ... */

/* get the record/column value after setting */
rc = SQLGetDescRec(hARD,
                    i,
                    colname,
                    sizeof(colname),
                    &namelen,
                    &type,
                    &subtype,
                    &width,
                    &precision,
                    &scale,
                    &nullable);
```

SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得

診断データ構造フィールドの現行値を返します。これは特定ハンドルに関連し、エラー、警告、および状況情報を含んでいます。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetDiagFieldW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLGetDiagField (
               SQLSMALLINT      HandleType,         /* fHandleType */
               SQLHANDLE        Handle,             /* hHandle */
               SQLSMALLINT      RecNumber,          /* iRecNumber */
               SQLSMALLINT      DiagIdentifier,     /* fDiagIdentifier */
               SQLPOINTER       DiagInfoPtr,        /* pDiagInfo */
               SQLSMALLINT      BufferLength,       /* cbDiagInfoMax */
               SQLSMALLINT      *StringLengthPtr);  /* *pcgDiagInfo */
```

関数引数

表 87. SQLGetDiagField 引数

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*HandleType*	入力	診断を必要とするハンドルのタイプを記述するハンドル・タイプ ID。以下のハンドル・タイプ ID があります。 • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
SQLHANDLE	*Handle*	入力	*HandleType* で示されるタイプの、診断データ構造のハンドル。
SQLSMALLINT	*RecNumber*	入力	アプリケーションによって検索される情報を収めた状況レコードを指定します。状況レコードは 1 から番号が付けられます。 *DiagIdentifier* 引数が診断ヘッダー・レコードのいずれかのフィールドを示す場合、*RecNumber* は 0 でなければなりません。0 でない場合、0 より大きい数でなければなりません。
SQLSMALLINT	*DiagIdentifier*	入力	値を戻す診断データ構造のフィールドを示します。詳しくは、DiagIdentifier 引数を参照してください。
SQLPOINTER	*DiagInfoPtr*	出力	診断情報を返すバッファーへのポインター。データ・タイプは、*DiagIdentifier* の値により異なります。

表 87. SQLGetDiagField 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER	*BufferLength*	入力	*DiagIdentifier* が ODBC 定義の診断である場合、次のようになります。 • *DiagInfoPtr* が文字ストリングまたはバイナリー・バッファーを指す場合、 *BufferLength* は **DiagInfoPtr* の長さでなければなりません。 • **DiagInfoPtr* が整数の場合、*BufferLength* は無視されます。 • **DiagInfoPtr* が Unicode ストリングの場合、*BufferLength* は偶数でなければなりません。 *DiagIdentifier* が CLI の診断である場合、次のようになります。 • **DiagInfoPtr* が文字ストリングを指すポインターの場合、*BufferLength* は、そのストリングを格納するのに必要なバイト数であるか、または SQL_NTS です。 • **DiagInfoPtr* がバイナリー・バッファーを指すポインターの場合、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *BufferLength* に入れます。それによって *BufferLength* は負の値になります。 • **DiagInfoPtr* が文字ストリングまたはバイナリー・ストリング以外の値を指すポインターの場合、*BufferLength* の値は SQL_IS_POINTER でなければなりません。 • **DiagInfoPtr* に固定長のデータ・タイプが入っている場合、 *BufferLength* は状況に応じて SQL_IS_INTEGER、SQL_IS_UINTEGER、SQL_IS_SMALLINT、または SQL_IS_USMALLINT になります。
SQLSMALLINT *	*StringLengthPtr*	出力	文字データの場合、**DiagInfoPtr* に戻すために使用できる SQLCHAR エレメントの合計数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの合計数) を戻すバッファーを指すポインター (NULL 終止符文字分のバイト数を除く)。戻り値として使用できるバイト数が *BufferLength* より大きい場合、**DiagInfoPtr* 内のテキストは *BufferLength* から NULL 終止符文字の長さを減算した長さに切り捨てられます。非文字データの場合、この引数は無視されます。

使用法

一般にアプリケーションは SQLGetDiagField() を呼び出して、以下の 3 つの目標の 1 つを成し遂げます。

1. 関数呼び出しで SQL_ERROR または SQL_SUCCESS_WITH_INFO (あるいは、SQLBrowseConnect() 関数で SQL_NEED_DATA) が戻されたときに特定のエラーまたは警告情報を得る。
2. SQLExecute()、SQLExecDirect()、SQLBulkOperations()、または SQLSetPos() の呼び出しで、挿入、削除、または更新操作が実行された時点で影響を受けたデータ・ソースの行数を検出する (SQL_DIAG_ROW_COUNT ヘッダー・フィールドから)。あるいは、現在開いている静的な両方向スクロール・カーソルにある行数を検出する (SQL_DIAG_CURSOR_ROW_COUNT ヘッダー・フィールドから)。
3. SQLExecDirect() または SQLExecute() への呼び出しで、どの関数が実行されたかを判別する (SQL_DIAG_DYNAMIC_FUNCTION および SQL_DIAG_DYNAMIC_FUNCTION_CODE ヘッダー・フィールドから)。

CLI 関数はいずれも、呼び出されるたびに 0 個以上のエラーを通知する可能性があるので、アプリケーションはどの関数呼び出しの後にも SQLGetDiagField() を呼び出すことができます。 SQLGetDiagField() は、 *Handle* 引数に指定された診断データ構造に最後に関連付けられた診断情報だけを取り出します。アプリケーションが別の関数を呼び出す場合、同一ハンドルによる直前の呼び出しの診断情報は失われます。

SQLGetDiagField() が SQL_SUCCESS を戻す限り、アプリケーションは *RecNumber* を増分することによってすべての診断記録をスキャンすることができます。状況記録の数は、SQL_DIAG_NUMBER ヘッダー・フィールドに示されます。
SQLGetDiagField() への呼び出しは、ヘッダーおよび状況レコードに関する限り非破壊です。 SQLGetDiagField()、SQLGetDiagRec()、または SQLError() 以外の他の関数を途中で呼び出した (この場合、同一ハンドルでのレコードを通知されます) のでない限り、アプリケーションは後で再び SQLGetDiagField() を呼び出してレコードからフィールドを取り出すことができます。

アプリケーションはいつでも SQLGetDiagField() を呼び出して何らかの診断フィールドを返すことができます。ただし SQL_DIAG_ROW_COUNT は例外であり、この場合、*Handle* が、実行された SQL ステートメントの基盤となったステートメント・ハンドルでなかった場合、 SQL_ERROR が戻されます。他の診断フィールドが未定義の場合、 SQLGetDiagField() への呼び出しで SQL_SUCCESS が戻されます (その他のエラーが生じなかった場合)。それから、未定義値がフィールドに対して戻されます。

HandleType 引数

それぞれのハンドル・タイプに、関連する診断情報を付けることができます。
HandleType 引数は、*Handle* のハンドル・タイプを表します。

すべてのハンドル・タイプ (環境、接続、ステートメント、および記述子) のヘッダー・フィールドおよびレコード・フィールドが戻されるわけではありません。フィールドが適用されないそれらのハンドルは、ヘッダー・フィールドおよびレコード・フィールドのセクションに示されます。

CLI 固有のヘッダー診断フィールドはいずれも環境ハンドルに関連付けられていないはずです。

DiagIdentifier 引数

この引数は、診断データ構造にある必須フィールドの ID を示します。*RecNumber* が 1 以上であれば、フィールド内のデータは関数で返される診断情報を記述します。 *RecNumber* が 0 であれば、フィールドは診断データ構造のヘッダー内にあるので、そのフィールドには診断情報 (特定情報ではない) を返した関数呼び出しに属するデータが入っています。詳細は、*DiagIdentifier* 引数のヘッダーおよびレコードのフィールドのリストを参照してください。

状況レコードの順序

状況レコードは、行番号および診断のタイプに基づいた順序に並べられます。

2 つ以上の状況レコードがある場合、そのレコードの順序はまず行番号で決められます。以下の規則は、行によるエラーの順序を決めるのに適用されます。

- どの行にも対応していないレコードは、SQL_NO_ROW_NUMBER が -1 に定義されているので、特定の行に対応しているレコードの前に表示されます。
- 行番号が不明のレコードは、SQL_ROW_NUMBER_UNKNOWN が -2 に定義されているので、他のすべてのレコードの前に表示されます。
- 特定の行に属するすべてのレコードの場合、レコードは SQL_DIAG_ROW_NUMBER フィールドにある値でソートされます。影響を受けた最初の行のすべてのエラーおよび警告がリストされ、それから、影響を受けた次の行のすべてのエラーおよび警告、以下同様に示されていきます。

各行の内部で、または 1 つの行に対応していないすべてのレコードの場合、または行番号が不明の場合には、リストされる最初のレコードは一連のソート規則を使用して決められます。最初のレコードの後、行に影響する他のレコードの順序は定義されていません。アプリケーションは、最初のレコードの後、エラーが警告に優先するとみなすことはできません。アプリケーションは、すべての診断データ構造をスキャンして、成功しなかった関数への呼び出しに関するすべての情報を得るようにしてください。

次の規則は、1 つの行の中で最初のレコードを決めるためのものです。一番高いランクのレコードは、最初のレコードです。

- **エラー**。エラーを記述する状況レコードは、一番高いランクです。次の規則は、エラーをソートするためのものです。
  - トランザクション障害、または他のすべてのレコードよりランクが高いトランザクション障害の疑いを示すレコード。
  - 2 つ以上のレコードが同じエラー条件を記述する場合、X/Open CLI 仕様 (クラス 03 から HZ まで) で定義される SQLSTATE は、ODBC 定義およびドライバー定義の SQLSTATE よりランクが上です。
- **インプリメンテーション定義の No Data 値**。 CLI No Data 値 (クラス 02) を記述する状況レコードは 2 番目に高いランクです。
- **警告**。警告 (クラス 01) を記述する状況レコードは最も低いランクです。複数のレコードが同じ警告条件を記述する場合、X/Open CLI 仕様で定義される SQLSTATE の警告は、ODBC およびドライバー定義の SQLSTATE よりランクが上です。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA

診断

SQLGetDiagField() は、自分自身のエラーの値を通知しません。次の戻り値を使用して、自身の実行結果を報告します。

- SQL_SUCCESS: 関数により、診断情報が正常に戻されました。
- SQL_SUCCESS_WITH_INFO: **DiagInfoPtr* は小さすぎて要求された診断フィールドを保持できなかったので、診断フィールド内のデータは切り捨てられました。切り捨てられたかどうかを判別するには、アプリケーションで、*BufferLength* と、**StringLengthPtr* に書き込まれる、実際に使用できるバイト数を比較する必要があります。
- SQL_INVALID_HANDLE: *HandleType* と *Handle* で示されたハンドルは有効なハンドルではありません。
- SQL_ERROR: 考えられる原因は、以下のとおりです。
  - *DiagIdentifier* 引数は、有効な値の 1 つではありませんでした。
  - *DiagIdentifier* 引数は、SQL_DIAG_CURSOR_ROW_COUNT、SQL_DIAG_DYNAMIC_FUNCTION、SQL_DIAG_DYNAMIC_FUNCTION_CODE、または SQL_DIAG_ROW_COUNT でした。しかし、 *Handle* はステートメント・ハンドルではありませんでした。
  - *DiagIdentifier* が診断レコードからのフィールドを示したとき、 *RecNumber* 引数が負か 0 でした。 *RecNumber* はヘッダー・フィールドには無視されました。
  - 要求された値は文字ストリングで、*BufferLength* はゼロ未満でした。
- SQL_NO_DATA: *RecNumber* が *Handle* で指定されたハンドル用の診断レコード数より大きな値になっていました。また、*Handle* で示されるハンドルの診断レコードがない場合は、この関数はすべての正の *RecNumber* に対しても SQL_NO_DATA を戻します。

制限

なし。

SQLGetDiagRec 関数 (CLI) - 診断レコードの複数フィールド設定の取得

エラー、警告、および状況情報が入っている診断レコードの複数のフィールドの現行値を戻します。

1 回の呼び出しに対して 1 つの診断フィールドを戻す SQLGetDiagField() とは異なり、 SQLGetDiagRec() は、SQLSTATE、ネイティブなエラー・コード、およびエラー・メッセージ・テキストなど、診断レコードで共通に使用されている複数のフィールドを戻します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetDiagRecW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLGetDiagRec (
               SQLSMALLINT       HandleType,       /* fHandleType */
               SQLHANDLE         Handle,           /* hHandle */
               SQLSMALLINT       RecNumber,        /* iRecNumber */
               SQLCHAR           *SQLState,        /* *pszSqlState */
               SQLINTEGER        *NativeErrorPtr,  /* *pfNativeError */
               SQLCHAR           *MessageText,     /* *pszErrorMsg */
               SQLSMALLINT       BufferLength,     /* cbErrorMsgMax */
               SQLSMALLINT       *TextLengthPtr);  /* *pcbErrorMsg */
```

関数引数

表 88. SQLGetDiagRec の引数

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*HandleType*	入力	診断するハンドルのタイプを記述するハンドル・タイプ ID。以下のハンドル・タイプ ID があります。 • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
SQLHANDLE	*Handle*	入力	*HandleType* で示されるタイプの、診断データ構造のハンドル。
SQLSMALLINT	*RecNumber*	入力	アプリケーションによって検索される情報を収めた状況レコードを指定します。状況レコードは、1 から数えます。
SQLCHAR *	*SQLState*	出力	診断レコード *RecNumber* に関する SQLSTATE コードについて 5 文字および NULL 終止符を戻すバッファーを指すポインター。先頭の 2 文字はクラス、続く 3 文字はサブクラスを表します。
SQLINTEGER *	*NativeErrorPtr*	出力	データ・ソースに特定のネイティブなエラー・コードを戻すバッファーを指すポインター。
SQLCHAR *	*MessageText*	出力	エラー・メッセージ・テキストを戻すバッファーを指すポインター。 SQLGetDiagRec() が戻すフィールドは、テキスト・ストリングに入ります。

表 88. SQLGetDiagRec の引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER	*BufferLength*	入力	*MessageText* バッファーを格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLSMALLINT *	*TextLengthPtr*	出力	**MessageText* に戻すために使用できる SQLCHAR エレメントの合計数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの合計数) を戻すバッファーを指すポインター (NULL 終止符文字を除く)。戻り値として使用できる SQLCHAR または SQLWCHAR エレメントの数が *BufferLength* より大きい場合、**MessageText* 内のエラー・メッセージ・テキストは *BufferLength* から NULL 終止符文字の長さを減算した長さに切り捨てられます。

使用法

CLI 関数への直前の呼び出しで SQL_SUCCESS 以外の値が戻されると、アプリケーションは通常 SQLGetDiagRec() を呼び出します。しかし、どの関数でも呼び出されるたびに 0 個以上のエラーを通知できるので、アプリケーションは任意の関数呼び出しの後に SQLGetDiagRec() を呼び出すことができます。アプリケーションは、SQLGetDiagRec() を何回も呼び出して、診断データ構造のレコードの一部またはすべてを戻すことができます。

SQLGetDiagRec() は、診断データ構造レコードの複数のフィールドの入った文字ストリングを戻します。

SQL_DIAG_MESSAGE_TEXT (戻りタイプ CHAR *)
エラーまたは警告に関する通知メッセージ。

SQL_DIAG_NATIVE (戻りタイプ SQLINTEGER)
ドライバー/データ・ソース指定の固有エラー・コード。固有のエラー・コードがなければ、ドライバーは 0 を返します。

SQL_DIAG_SQLSTATE (戻りタイプ CHAR *)
5 文字の SQLSTATE 診断コード。

SQLGetDiagRec() は、診断データ構造のヘッダーからフィールドを戻すことはできません (*RecNumber* 引数が 0 より大きい値でなければならない)。これを行うには、アプリケーションは SQLGetDiagField() を呼び出す必要があります。

SQLGetDiagRec() は、 *Handle* 引数で指定されているハンドルに関連する最新の診断情報だけを取り出します。アプリケーションが SQLGetDiagRec() または SQLGetDiagField() 以外の別の関数を呼び出すと、直前の呼び出しで同じハンドルで検索した診断情報は失われます。

SQLGetDiagRec() が SQL_SUCCESS を戻す場合、ループ、つまり *RecNumber* を増分すればすべての診断レコードをスキャンできます。 SQLGetDiagRec() を呼び出しても、ヘッダーとレコード・フィールドは破壊されません。アプリケーションは、SQLGetDiagRec() または SQLGetDiagField() 以外の関数を途中で呼び出さないかぎり、あとでもう一度 SQLGetDiagRec() を呼び出してレコードからフィールドを検索

することができます。また、SQLGetDiagField() を呼び出せば、使用できる診断レコードの合計数である SQL_DIAG_NUMBER フィールドの値を取り出すことができます。その後、必要な回数だけ SQLGetDiagRec() を呼び出す必要があります。

HandleType 引数

それぞれのハンドル・タイプに、関連する診断情報を付けることができます。*HandleType* 引数は、*Handle* のハンドル・タイプを表します。

すべてのハンドル・タイプ (環境、接続、ステートメント、および記述子) のヘッダー・フィールドおよびレコード・フィールドが戻されるわけではありません。フィールドが適用されないそれらのハンドルは、 *DiagIdentifier* 引数のヘッダー・フィールドとレコード・フィールドのリスト内に示されています。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

SQLGetDiagRec() は、自分でエラー値を通知することはできません。次の戻り値を使用して、自身の実行結果を報告します。

- SQL_SUCCESS: 関数により、診断情報が正常に戻されました。
- SQL_SUCCESS_WITH_INFO: **MessageText* バッファーが小さすぎて、要求された診断メッセージが入りませんでした。診断レコードは生成されませんでした。切り捨てられたかどうかを判別するには、アプリケーションで、*BufferLength* と、**StringLengthPtr* に書き込まれる、実際に使用できるバイト数を比較する必要があります。
- SQL_INVALID_HANDLE: *HandleType* と *Handle* で示されたハンドルは有効なハンドルではありません。
- SQL_ERROR: 考えられる原因は、以下のとおりです。
  - *RecNumber* が負の値または 0 でした。
  - *BufferLength* が 0 未満でした。
- SQL_NO_DATA: *RecNumber* が *Handle* で指定されたハンドル用の診断レコード数より大きな値になっていました。また、*Handle* で示されるハンドルの診断レコードがない場合は、この関数はすべての正の *RecNumber* に対しても SQL_NO_DATA を戻します。

例

```
/* get multiple fields settings of diagnostic record */
SQLGetDiagRec(SQL_HANDLE_STMT,
              hstmt,
              1,
              sqlstate,
              &sqlcode,
              message,
              200,
              &length);
```

SQLGetEnvAttr 関数 (CLI) - 現行の環境属性値の検索

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() 関数を使用して設定されます。

仕様:

- **CLI** 2.1
- **ISO CLI**

構文

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

関数引数

表 89. SQLGetEnvAttr 引数

データ・タイプ	引数	使用法	説明
SQLHENV	*EnvironmentHandle*	入力	環境ハンドル。
SQLINTEGER	*Attribute*	入力	受け取る属性。環境属性とその説明のリストについては、環境属性とその説明の項を参照してください。
SQLPOINTER	*ValuePtr*	出力	*属性* によって指定されている属性の現行値を返すメモリーを指すポインター。
SQLINTEGER	*BufferLength*	入力	属性値が文字ストリングの場合は *ValuePtr* が指すバッファーの最大サイズ。それ以外の場合は無視されます。
SQLINTEGER *	*StringLengthPtr*	出力	*ValuePtr* に戻すために使用できる総バイト数 (NULL 終止符文字に戻されるバイト数を除く) を戻すバッファーを指すポインター。 *ValuePtr* が NULL ポインターである場合、長さは戻されません。属性値が文字ストリングであって、戻りに使用できるバイト数が *BufferLength* 以上の場合、 *ValuePtr* のデータは、*BufferLength* から NULL 終止符文字の長さを減算した長さに切り捨てられ、 CLI によってヌル終了になります。

Attribute がストリングを示さない場合、CLI は *BufferLength* を無視し、*StringLengthPtr* を設定しません。

使用法

環境ハンドルの割り振りから解放までの間にいつでも SQLGetEnvAttr() を呼び出すことができます。この関数は、環境属性の現行値を取得します。

戻りコード

- SQL_SUCCESS
- SQL_ERROR

- SQL_INVALID_HANDLE

診断

表 90. SQLGetEnvAttr SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY092	オプション・タイプが範囲外です。	無効な *Attribute* 値を指定しました。

制限

なし。

例

```
/* retrieve the current environment attribute value */
cliRC = SQLGetEnvAttr(henv, SQL_ATTR_OUTPUT_NTS, &output_nts, 0, NULL);
```

SQLGetFunctions 関数 (CLI) - 関数の取得

特定の CLI または ODBC 関数がサポートされているかどうかを判別します。

これでアプリケーションは、さまざまなデータベース・サーバーに接続するときに、さまざまなサポート・レベルに適応することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0
- **ISO CLI**

この関数を呼び出す前に、データベース・サーバーへの接続が存在していることが必要です。

構文

```
SQLRETURN   SQLGetFunctions  (
               SQLHDBC          ConnectionHandle,  /* hdbc */
               SQLUSMALLINT     FunctionId,        /* fFunction */
               SQLUSMALLINT     *SupportedPtr);    /* pfExists */
```

関数引数

表 91. SQLGetFunctions 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	データベース接続ハンドル。
SQLUSMALLINT	*FunctionId*	入力	照会される関数。

表 91. SQLGetFunctions 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLUSMALLINT *	*SupportedPtr*	出力	照会される関数がサポートされるかどうかに基づいて、この関数が SQL_TRUE または SQL_FALSE を戻すロケーションを指すポインター。

使用法

FunctionId が SQL_API_ALL_FUNCTIONS に設定されている場合、 *SupportedPtr* は 100 個のエレメントの SQLSMALLINT 配列を指していなければなりません。多くの関数を識別するのに使われる *FunctionId* 値を使ってこの配列に指標が付けられます。この配列の一部のエレメントは使用されておらず、予約済みです。 100 より大きい値になっている *FunctionId* もあるので、関数のリストを取得するために配列方式を使うことはできません。 100 以上の値を持つすべての *SQLGetFunctions()* 値には、 SQLGetFunction() 呼び出しを明示的に出す必要があります。 *FunctionId* 値の完全セットは `sqlcli1.h` に定義されています。

注: LOB サポート関数 SQLGetLength()、 SQLGetPosition()、 SQLGetSubString()、 SQLBindFileToCol()、 SQLBindFileToCol() は、 LOB データ・タイプをサポートしない IBM RDBMS に接続している場合にはサポートされません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 92. SQLGetFunctions SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	データベース接続が確立する前に、SQLGetFunctions() が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。

許可

なし。

例

```
/* check to see if SQLGetInfo() is supported */
cliRC = SQLGetFunctions(hdbc, SQL_API_SQLGETINFO, &supported);
```

参照

なし。

SQLGetInfo 関数 (CLI) - 一般情報の取得

アプリケーションが現在接続している DBMS に関する一般情報を戻します。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

SQLGetInfo() は、アプリケーションが現在接続しているデータベース管理システム (DBMS) に関する一般情報を戻します。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetInfoW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN   SQLGetInfo       (
               SQLHDBC           ConnectionHandle,   /* hdbc */
               SQLUSMALLINT      InfoType,           /* fInfoType */
               SQLPOINTER        InfoValuePtr,       /* rgbInfoValue */
               SQLSMALLINT       BufferLength,       /* cbInfoValueMax */
               SQLSMALLINT       *StringLengthPtr);  /* pcbInfoValue */
```

関数引数

表 93. SQLGetInfo 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	データベース接続ハンドル。
SQLUSMALLINT	*InfoType*	入力	必要な情報のタイプ。この引数に指定できる値は、SQLGetInfo() から戻される情報に説明されています。

表 93. SQLGetInfo 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLPOINTER	*InfoValuePtr*	出力および入力	必要な情報がこの関数によって保管されるバッファーを指すポインター。検索される情報のタイプに基づいて、以下の 5 つの情報タイプを戻すことができます。 • 16 ビットの整数値 • 32 ビットの整数値 • 32 ビットのバイナリー数値 • 32 ビット・マスク • ヌル終了文字ストリング *InfoType* 引数が SQL_DRIVER_HDESC または SQL_DRIVER_HSTMT である場合、 *InfoValuePtr* は入力引数にも出力引数にもなります。
SQLSMALLINT	*BufferLength*	入力	*InfoValuePtr* ポインターで指示されるバッファーの最大長。**InfoValuePtr* が Unicode ストリングの場合、*BufferLength* 引数は偶数でなければなりません。
SQLSMALLINT *	*StringLengthPtr*	出力	戻りに使用できる情報の合計バイト数をこの関数が戻すロケーションを指すポインター。ストリング出力の場合、長さにはヌル終了文字は含まれません。 *StringLengthPtr* が指すロケーションの値が *BufferLength* に指定されているサイズより大きい場合、ストリング出力情報が *BufferLength* - 1 バイトに切り捨てられ、この関数は SQL_SUCCESS_WITH_INFO を戻します。

使用法

InfoType 引数の有効値のリストと、その値に関して `SQLGetInfo()` 関数が戻す情報の詳細は、SQLGetInfo() から戻される情報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 94. SQLGetInfo SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	要求された情報がストリングとして戻され、このストリングの長さは *BufferLength* 引数に指定されているアプリケーション・バッファーの長さを超過しています。*StringLengthPtr* 引数には、要求された情報の実際の (切り捨てられていない) 長さが含まれています。(関数は、SQL_SUCCESS_WITH_INFO 戻りコードを返します。)

表 94. SQLGetInfo SQLSTATE (続き)

SQLSTATE	説明	解説
08003	接続がクローズされています。	*InfoType* 引数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 情報だけです。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY090	ストリングまたはバッファーの長さが無効です。	*BufferLength* 引数に指定された値は 0 より小さい値です。
HY096	情報タイプが範囲外です。	無効な *InfoType* 引数が指定されています。
HYC00	ドライバーが使用できません。	*InfoType* 引数に指定された値は、CLI とデータ・ソースのどちらかでサポートされていません。

制限

なし。

例

```
/* get server name information */
cliRC = SQLGetInfo(hdbc, SQL_DBMS_NAME, imageInfoBuf, 255, &outlen);

/* ... */

/* get client driver name information */
cliRC = SQLGetInfo(hdbc, SQL_DRIVER_NAME, imageInfoBuf, 255, &outlen);
```

SQLGetInfo() から戻される情報

注: CLI は、この表の *InfoType* 引数ごとに値を返します。*InfoType* 引数が適用されていなかったりサポートされていない場合、結果は戻りタイプに従属します。戻りタイプと結果の関係は以下のとおりです。

- 文字ストリング ("Y" または "N") の場合、"N" が戻されます。
- 文字ストリング ("Y" でも "N" でもない) の場合、空ストリングが戻されます。
- 32 ビット整数の場合、0 (ゼロ) が戻されます。
- 32 ビット・マスクの場合、0 (ゼロ) が戻されます。

SQL_ACCESSIBLE_PROCEDURES (ストリング)
文字ストリング "Y" は、関数 SQLProcedures() で返されるプロシージャーをすべて実行できることを示します。"N" は、実行できないプロシージャーが返されている可能性があることを示します。

SQL_ACCESSIBLE_TABLES (ストリング)
文字ストリング "Y" は、関数 SQLTables() で返されるすべての表に対する SELECT 特権が確保されることを示します。"N" は、アクセスできない表が返されている可能性があることを示します。

SQL_AGGREGATE_FUNCTIONS (32 ビット・マスク)
下記の集約関数に関するサポートを列挙するビット・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_ALTER_DOMAIN (32 ビット・マスク)
CLI は 0 を返し、ALTER DOMAIN ステートメントがサポートされていないことを示します。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_AD_ADD_CONSTRAINT_DEFERRABLE
- SQL_AD_ADD_CONSTRAINT_NON_DEFERRABLE
- SQL_AD_ADD_CONSTRAINT_INITIALLY_DEFERRED
- SQL_AD_ADD_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_AD_ADD_DOMAIN_CONSTRAINT
- SQL_AD_ADD_DOMAIN_DEFAULT
- SQL_AD_CONSTRAINT_NAME_DEFINITION
- SQL_AD_DROP_DOMAIN_CONSTRAINT
- SQL_AD_DROP_DOMAIN_DEFAULT

SQL_ALTER_TABLE (32 ビット・マスク)
ALTER TABLE ステートメント中のどの節を DBMS がサポートしているかを示します。
- SQL_AT_ADD_COLUMN_COLLATION
- SQL_AT_ADD_COLUMN_DEFAULT
- SQL_AT_ADD_COLUMN_SINGLE
- SQL_AT_ADD_CONSTRAINT
- SQL_AT_ADD_TABLE_CONSTRAINT
- SQL_AT_CONSTRAINT_NAME_DEFINITION
- SQL_AT_DROP_COLUMN_CASCADE
- SQL_AT_DROP_COLUMN_DEFAULT
- SQL_AT_DROP_COLUMN_RESTRICT
- SQL_AT_DROP_TABLE_CONSTRAINT_CASCADE
- SQL_AT_DROP_TABLE_CONSTRAINT_RESTRICT
- SQL_AT_SET_COLUMN_DEFAULT
- SQL_AT_CONSTRAINT_INITIALLY_DEFERRED
- SQL_AT_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_AT_CONSTRAINT_DEFERRABLE
- SQL_AT_CONSTRAINT_NON_DEFERRABLE

SQL_APPLICATION_CODEPAGE (32 ビット符号なし整数)
アプリケーション・コード・ページ。

SQL_ASYNC_MODE (32 ビット符号なし整数)
ドライバーにおける非同期サポートのレベルを示します。

- SQL_AM_CONNECTION : 非同期実行がサポートされる接続レベル。特定の接続ハンドルに関連付けられているすべてのステートメント・ハンドルが非同期モードになっているか、逆にすべてが同期モードになっています。接続上のステートメント・ハンドルは、同一の接続上にある別のステートメント・ハンドルが同期モードになっていると、非同期モードにすることができません (その逆も同様)。
- SQL_AM_STATEMENT : 非同期実行がサポートされるステートメント・レベル。接続ハンドルに関連付けられたステートメント・ハンドルを、同一の接続上の他のステートメント・ハンドルが同期モードであっても、非同期モードにすることができます。
- SQL_AM_NONE : 非同期モードはサポートされません。

  CLI/ODBC の構成キーワード ASYNCENABLE が非同期実行を無効にする設定になっている場合にも、この値が返されます。

SQL_BATCH_ROW_COUNT (32 ビット・マスク)

行カウントの処理方法が示されます。 CLI は常に SQL_BRC_ROLLED_UP を返し、連続した INSERT、DELETE、または UPDATE の各ステートメントの行カウントが 1 つにされることを示します。

ODBC はさらに、CLI が返さない以下の値を定義します。

- SQL_BRC_PROCEDURES
- SQL_BRC_EXPLICIT

SQL_BATCH_SUPPORT (32 ビット・マスク)

サポートされているバッチのレベルを示します。

- SQL_BS_SELECT_EXPLICIT : これは、結果セットを生成するステートメントを指定できる明示バッチをサポートします。
- SQL_BS_ROW_COUNT_EXPLICIT : これは、行カウントを生成するステートメントを指定できる明示バッチをサポートします。
- SQL_BS_SELECT_PROC : これは、結果セットを生成するステートメントを指定できる明示プロシージャーをサポートします。
- SQL_BS_ROW_COUNT_PROC : これは、行カウントを生成するステートメントを指定できる明示プロシージャーをサポートします。

SQL_BOOKMARK_PERSISTENCE (32 ビット・マスク)

演算後もブックマークが有効のままになる場合を示します。

- SQL_BP_CLOSE : アプリケーションが SQL_CLOSE オプションを指定した `SQLFreeStmt()`、または `SQLCloseCursor()` を呼び出して、ステートメントに関連したカーソルをクローズした後。
- SQL_BP_DELETE : 当該行が削除された後。
- SQL_BP_DROP : ブックマークが有効なのは、アプリケーションが `SQLFreeHandle()` を SQL_HANDLE_STMT の *HandleType* とともに呼び出して、ステートメントをドロップした後です。
- SQL_BP_TRANSACTION : アプリケーションがトランザクションをコミットまたはロールバックした後。
- SQL_BP_UPDATE : その行のいずれかの列 (キー列を含む) が更新された後。
- SQL_BP_OTHER_HSTMT : あるステートメントに関連付けられたブックマークを、別のステートメントで使用できます。SQL_BP_CLOSE または

SQL_BP_DROP が指定されていない限り、最初のステートメント上のカーソルがオープンしていなければなりません。

SQL_CATALOG_LOCATION (16 ビット整数)
修飾表名中の修飾子の位置を示す 16 ビット整数値。CLI では、この情報タイプについて常に SQL_CL_START が返されます。 ODBC は、CLI から戻されない値 SQL_CL_END も定義します。

CLI の前のバージョンでは、この *InfoType* は SQL_QUALIFIER_LOCATION でした。

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、カタログ名をサポートしていないことを示します。

SQL_CATALOG_NAME_SEPARATOR (ストリング)
カタログ名と、その後またはその前に付く修飾名エレメントを区切る区切り記号として使用される文字。

CLI の前のバージョンでは、この *InfoType* は SQL_QUALIFIER_NAME_SEPARATOR でした。

SQL_CATALOG_TERM (ストリング)
データベース・ベンダーの修飾子 (カタログ) 用の用語。

ベンダーが、3 部分から成る名前の高位の部分に使用する名前。

ターゲット DBMS では 3 つの部分から成る名前がサポートされていない場合、長さゼロのストリングが戻されます。

CLI の前のバージョンでは、この *InfoType* は SQL_QUALIFIER_TERM でした。

SQL_CATALOG_USAGE (32 ビット・マスク)

カタログを使用でき、ステートメントを列挙する 32 ビット・マスクは次のとおりです。SQL_CATALOG_USAGE は、カタログ固有であることを除き、SQL_SCHEMA_USAGE に似ています。

- SQL_CU_DML_STATEMENTS : すべてのデータ操作言語 (DML) ステートメント。
- SQL_CU_INDEX_DEFINITION : すべての索引定義ステートメント。
- SQL_CU_PRIVILEGE_DEFINITION : すべての特権定義ステートメント。
- SQL_CU_PROCEDURE_INVOCATION : ODBC プロシージャー呼び出しステートメント。
- SQL_CU_TABLE_DEFINITION : すべての表定義ステートメント。

データ・ソースでカタログがサポートされていない場合、0 の値が戻されます。

CLI の前のバージョンでは、この *InfoType* 引数は SQL_QUALIFIER_USAGE でした。

SQL_COLLATION_SEQ (ストリング)
このサーバーのデフォルト文字セットにおけるデフォルトの照合シーケンスの名前 (例えば ISO 8859-1 または EBCDIC) を示します。照合シーケンスが不明であると、空ストリングが返されます。

SQL_COLUMN_ALIAS (ストリング)
列別名がサポートされている場合は "Y" が返され、サポートされていない場合は "N" が返されます。

SQL_CONCAT_NULL_BEHAVIOR (16 ビット整数)
NULL 値の文字データ・タイプの列と非 NULL 値の文字データ・タイプの列の連結をどのように処理するかを示します。
- SQL_CB_NULL : NULL 値 (この動作は IBM RDBMS の場合)。
- SQL_CB_NON_NULL : 非 NULL 列値が連結されたもの。

SQL_CONVERT_* (32 ビット・マスク)
SQL_CONVERT_BIGINT (32 ビット・マスク)
SQL_CONVERT_BINARY (32 ビット・マスク)
SQL_CONVERT_BIT (32 ビット・マスク)
SQL_CONVERT_CHAR (32 ビット・マスク)
SQL_CONVERT_DATE (32 ビット・マスク)
SQL_CONVERT_DECIMAL (32 ビット・マスク)
SQL_CONVERT_DOUBLE (32 ビット・マスク)
SQL_CONVERT_FLOAT (32 ビット・マスク)
SQL_CONVERT_INTEGER (32 ビット・マスク)
SQL_CONVERT_INTERVAL_YEAR_MONTH (32 ビット・マスク)
SQL_CONVERT_INTERVAL_DAY_TIME (32 ビット・マスク)
SQL_CONVERT_LONGVARBINARY (32 ビット・マスク)
SQL_CONVERT_LONGVARCHAR (32 ビット・マスク)
SQL_CONVERT_NUMERIC (32 ビット・マスク)
SQL_CONVERT_REAL (32 ビット・マスク)
SQL_CONVERT_SMALLINT (32 ビット・マスク)
SQL_CONVERT_TIME (32 ビット・マスク)
SQL_CONVERT_TIMESTAMP (32 ビット・マスク)
SQL_CONVERT_TINYINT (32 ビット・マスク)
SQL_CONVERT_VARBINARY (32 ビット・マスク)
SQL_CONVERT_VARCHAR (32 ビット・マスク)
SQL_CONVERT_WCHAR (32 ビット・マスク)
SQL_CONVERT_WLONGVARCHAR (32 ビット・マスク)
SQL_CONVERT_WVARCHAR (32 ビット・マスク)

CONVERT スカラー関数を用いて行われる、*InfoType* に名前指定されているタイプのデータの変換で、データ・ソースによってサポートされているものを示しています。ビット・マスクがゼロと等しい場合、データ・ソースは、同じデータ・タイプへの変換を含め、名前のあげられているデータ・タイプの変換をサポートしません。
例えば、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しているかどうかを調べるため、アプリケーションは、SQL_CONVERT_INTEGER の *InfoType* 引数とともに SQLGetInfo() 関数を呼び出します。その後アプリケーションは、戻されたビット・マスクに対して SQL_CVT_DECIMAL との AND 演算を実行します。結果値が非ゼロであれば、変換はサポートされています。
以下のビット・マスクを用いて、どの変換がサポートされているかを判別します。
- SQL_CVT_BIGINT

- SQL_CVT_BINARY
- SQL_CVT_BIT
- SQL_CVT_CHAR
- SQL_CVT_DATE
- SQL_CVT_DECIMAL
- SQL_CVT_DOUBLE
- SQL_CVT_FLOAT
- SQL_CVT_INTEGER
- SQL_CVT_INTERVAL_YEAR_MONTH
- SQL_CVT_INTERVAL_DAY_TIME
- SQL_CVT_LONGVARBINARY
- SQL_CVT_LONGVARCHAR
- SQL_CVT_NUMERIC
- SQL_CVT_REAL
- SQL_CVT_SMALLINT
- SQL_CVT_TIME
- SQL_CVT_TIMESTAMP
- SQL_CVT_TINYINT
- SQL_CVT_VARBINARY
- SQL_CVT_VARCHAR
- SQL_CVT_WCHAR
- SQL_CVT_WLONGVARCHAR
- SQL_CVT_WVARCHAR

SQL_CONNECT_CODEPAGE (32 ビット符号なし整数)
現在の接続のコード・ページ。

SQL_CONVERT_FUNCTIONS (32 ビット・マスク)
ドライバーおよび関連データ・ソースによってサポートされているスカラー変換関数を示しています。

CLI バージョン 2.1.1 およびそれ以降では、char 変数 (CHAR、VARCHAR、LONG VARCHAR および CLOB) と DOUBLE (または FLOAT) との間における ODBC スカラー変換をサポートしています。
- 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します。

SQL_CORRELATION_NAME (16 ビット整数)
サーバーでサポートされる相関名の程度を示します。
- SQL_CN_ANY : すべての有効なユーザー定義名がサポートされます。
- SQL_CN_NONE : 相関名はサポートされていません。
- SQL_CN_DIFFERENT : 相関名はサポートされていますが、その名前が表している表の名前と違う名前でなければなりません。

SQL_CREATE_ASSERTION (32 ビット・マスク)
CREATE ASSERTION ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、CREATE ASSERTION ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_CA_CREATE_ASSERTION
- SQL_CA_CONSTRAINT_INITIALLY_DEFERRED

- SQL_CA_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_CA_CONSTRAINT_DEFERRABLE
- SQL_CA_CONSTRAINT_NON_DEFERRABLE

SQL_CREATE_CHARACTER_SET (32 ビット・マスク)
CREATE CHARACTER SET ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、CREATE CHARACTER SET ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_CCS_CREATE_CHARACTER_SET
- SQL_CCS_COLLATE_CLAUSE
- SQL_CCS_LIMITED_COLLATION

SQL_CREATE_COLLATION (32 ビット・マスク)
CREATE COLLATION ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、CREATE COLLATION ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_CCOL_CREATE_COLLATION

SQL_CREATE_DOMAIN (32 ビット・マスク)
CREATE DOMAIN ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、CREATE DOMAIN ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_CDO_CREATE_DOMAIN
- SQL_CDO_CONSTRAINT_NAME_DEFINITION
- SQL_CDO_DEFAULT
- SQL_CDO_CONSTRAINT
- SQL_CDO_COLLATION
- SQL_CDO_CONSTRAINT_INITIALLY_DEFERRED
- SQL_CDO_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_CDO_CONSTRAINT_DEFERRABLE
- SQL_CDO_CONSTRAINT_NON_DEFERRABLE

SQL_CREATE_MODULE (32 ビット・マスク)
CREATE MODULE ステートメント内のどの節が DBMS でサポートされているかを示します。DB2 for z/OS の場合、CLI は常にゼロを戻します。

CLI は、次の値を戻します。
- SQL_CM_CREATE_MODULE
- SQL_CM_AUTHORIZATION
- SQL_CM_DEFAULT_CHARACTER_SET

SQL_CREATE_SCHEMA (32 ビット・マスク)
CREATE SCHEMA ステートメント中のどの節を DBMS がサポートしているかを示します。
- SQL_CS_CREATE_SCHEMA
- SQL_CS_AUTHORIZATION
- SQL_CS_DEFAULT_CHARACTER_SET

SQL_CREATE_TABLE (32 ビット・マスク)
CREATE TABLE ステートメント中のどの節を DBMS がサポートしているかを示します。

以下のビット・マスクを用いて、どの節がサポートされているかを示します。
- SQL_CT_CREATE_TABLE
- SQL_CT_TABLE_CONSTRAINT
- SQL_CT_CONSTRAINT_NAME_DEFINITION

以下のビットは、一時表を作成する能力を指定します。
- SQL_CT_COMMIT_PRESERVE : 削除行はコミット時に保存されます。
- SQL_CT_COMMIT_DELETE : 削除行はコミット時に削除されます。
- SQL_CT_GLOBAL_TEMPORARY : グローバル一時表を作成できます。
- SQL_CT_LOCAL_TEMPORARY : ローカル一時表を作成できます。

以下のビットは、列制約を作成する能力を指定します。
- SQL_CT_COLUMN_CONSTRAINT : 列制約の指定がサポートされます。
- SQL_CT_COLUMN_DEFAULT : 列のデフォルト値の指定がサポートされます。
- SQL_CT_COLUMN_COLLATION : 列照合の指定がサポートされます。

以下のビットは、列または表の制約の指定がサポートされている場合に、サポートする制約属性を指定します。
- SQL_CT_CONSTRAINT_INITIALLY_DEFERRED
- SQL_CT_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_CT_CONSTRAINT_DEFERRABLE
- SQL_CT_CONSTRAINT_NON_DEFERRABLE

SQL_CREATE_TRANSLATION (32 ビット・マスク)
CREATE TRANSLATION ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、CREATE TRANSLATION ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_CTR_CREATE_TRANSLATION

SQL_CREATE_VIEW (32 ビット・マスク)
CREATE VIEW ステートメント中のどの節を DBMS がサポートしているかを示します。
- SQL_CV_CREATE_VIEW
- SQL_CV_CHECK_OPTION
- SQL_CV_CASCADED
- SQL_CV_LOCAL

戻り値 0 は、CREATE VIEW ステートメントがサポートされていないことを意味します。

SQL_CURSOR_COMMIT_BEHAVIOR (16 ビット整数)
COMMIT 操作がカーソルにどのような影響を与えるかを示します。値は、以下のとおりです。
- SQL_CB_DELETE。カーソルは削除され、動的 SQL ステートメントに関するアクセス・プランがドロップされます。

- SQL_CB_CLOSE。カーソルは削除されますが、動的 SQL ステートメントに関するアクセス・プランは保持されます (非照会ステートメントを含む)。
- SQL_CB_PRESERVE。カーソルと、動的ステートメントに関するアクセス・プランは保持されます (非照会ステートメントを含む)。アプリケーションが継続してデータをフェッチするか、またはカーソルをクローズしてステートメントを準備せずに照会を再実行できます。

  注: COMMIT の後で、定位置更新や削除などのアクションを行うには、その前に FETCH を発行してカーソルを再配置しなければなりません。

SQL_CURSOR_ROLLBACK_BEHAVIOR (16 ビット整数)
ROLLBACK 操作がどのようにカーソルに影響を与えるかを示します。値は、以下のとおりです。

- SQL_CB_DELETE。カーソルは削除され、動的 SQL ステートメントに関するアクセス・プランがドロップされます。
- SQL_CB_CLOSE。カーソルは削除されますが、動的 SQL ステートメントに関するアクセス・プランは保持されます (非照会ステートメントを含む)。
- SQL_CB_PRESERVE。カーソルと、動的ステートメントに関するアクセス・プランは保持されます (非照会ステートメントを含む)。アプリケーションが継続してデータをフェッチするか、またはカーソルをクローズしてステートメントを準備せずに照会を再実行できます。

  注: DB2 サーバーには SQL_CB_PRESERVE 特性がありません。

SQL_CURSOR_SENSITIVITY (32 ビット符号なし整数)
以下のように、カーソル・センシティビティーのサポートを示します。

- SQL_INSENSITIVE。ステートメント・ハンドル上のすべてのカーソルが示す結果セットには、同一のトランザクション内にある別のカーソルがその結果セットに対して行った変更が反映されません。
- SQL_UNSPECIFIED。同一のトランザクション内の別のカーソルが結果セットに加えた変更を、ステートメント・ハンドル上のカーソルが可視にするかどうかを指定しません。ステートメント・ハンドル上のカーソルは、そのような変更をどれも可視にしないか、その一部、あるいは全部を可視にすることができます。
- SQL_SENSITIVE。カーソルは、同一のトランザクション内の別のカーソルによる変更を感知します。

SQL_DATA_SOURCE_NAME (ストリング)
接続中に使用されるデータ・ソース名を示します。アプリケーションが `SQLConnect()` を呼び出した場合、この文字ストリングが *szDSN* 引数の値になります。アプリケーションが `SQLDriverConnect()` または `SQLBrowseConnect()` を呼び出した場合、この文字ストリングがドライバーに渡される接続ストリング内の DSN キーワードの値になります。接続ストリング内に DSN キーワードが含まれていなかった場合、この文字ストリングは空ストリングになります。

SQL_DATA_SOURCE_READ_ONLY (ストリング)
文字ストリング "Y" は、データベースを READ ONLY モードに設定することを示し、 "N" は、READ ONLY モードにセットしないことを示しま

す。この特性はデータ・ソースそのものにのみ帰属する特性であって、データ・ソースにアクセスするのに使用できるドライバーの特性ではありません。

SQL_DATABASE_CODEPAGE (32 ビット符号なし整数)
アプリケーションが現在接続している先のデータベースのコード・ページ。

SQL_DATABASE_NAME (ストリング)
現在使用中のデータベースの名前。

注: このストリングは、非ホスト・システムで SELECT CURRENT SERVER ステートメントによって戻されるものと同じです。 DB2 for z/OS や DB2 for i などのホスト・データベースの場合、戻されるストリングは、DCS データベース名になります。このデータベース名は、DB2 Connect ゲートウェイでの CATALOG DCS DATABASE DIRECTORY コマンドの発行時に提供されています。

SQL_DATETIME_LITERALS (32 ビット符号なし整数)
DBMS がサポートする、日時リテラルを示します。 CLI は常にゼロを返し、日時リテラルをサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_DL_SQL92_DATE
- SQL_DL_SQL92_TIME
- SQL_DL_SQL92_TIMESTAMP
- SQL_DL_SQL92_INTERVAL_YEAR
- SQL_DL_SQL92_INTERVAL_MONTH
- SQL_DL_SQL92_INTERVAL_DAY
- SQL_DL_SQL92_INTERVAL_HOUR
- SQL_DL_SQL92_INTERVAL_MINUTE
- SQL_DL_SQL92_INTERVAL_SECOND
- SQL_DL_SQL92_INTERVAL_YEAR_TO_MONTH
- SQL_DL_SQL92_INTERVAL_DAY_TO_HOUR
- SQL_DL_SQL92_INTERVAL_DAY_TO_MINUTE
- SQL_DL_SQL92_INTERVAL_DAY_TO_SECOND
- SQL_DL_SQL92_INTERVAL_HOUR_TO_MINUTE
- SQL_DL_SQL92_INTERVAL_HOUR_TO_SECOND
- SQL_DL_SQL92_INTERVAL_MINUTE_TO_SECOND

SQL_DBMS_NAME (ストリング)
アクセスされる DBMS 製品の名前。

例:
- "DB2/6000"
- "DB2/2"

SQL_DBMS_VER (ストリング)
アクセスされる DBMS 製品のバージョン。書式 'mm.vv.rrrr' のストリング (*mm* は主バージョン、*vv* は副バージョン、および *rrrr* はリリース番号)。たとえば、"0r.01.0000" は主バージョン r、副バージョン 1、リリース 0 に変換します。

SQL_DDL_INDEX (32 ビット符号なし整数)
索引の作成とドロップのサポートを示します。
- SQL_DI_CREATE_INDEX
- SQL_DI_DROP_INDEX

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

以下のいずれかのマスクが返されます。

- SQL_TXN_READ_UNCOMMITTED : 変更が行われると、すべてのトランザクションがただちにそれを認識します (ダーティー読み取り、反復不能読み取り、および幻像読み取りが可能です)。

  この動作は、IBM データベースの非コミット読み取りレベルと同等です。
- SQL_TXN_READ_COMMITTED : トランザクション 1 による行読み取りを、トランザクション 2 によって変更およびコミットすることができます (反復不能読み取りおよび幻像読み取りが可能です)。

  この動作は、IBM データベースのカーソル固定レベルと同等です。
- SQL_TXN_REPEATABLE_READ : トランザクションは、検索条件に一致する行またはペンディング中のトランザクションを追加したり除去したりできます (非コミット読み取りと幻像読み取りが可能です)。

  この動作は、IBM データベースの読み取り固定レベルと同等です。
- SQL_TXN_SERIALIZABLE : ペンディング中のトランザクションによる影響を受けたデータは他のトランザクションで使用できません (反復可能読み取りも、幻像読み取りも不可能です)。

  この動作は、IBM データベースの反復可能読み取りレベルと同等です。
- SQL_TXN_VERSIONING : IBM DBMS には適用されません。
- SQL_TXN_NOCOMMIT : すべての変更内容は操作が正常に終了した時点で有効にコミットされます。明示コミットやロールバックはできません。

  これは IBM DB2 for IBM i の分離レベルです。

IBM の用語では、
- SQL_TXN_READ_UNCOMMITTED は、非コミット読み取りです。
- SQL_TXN_READ_COMMITTED は、カーソル固定です。
- SQL_TXN_REPEATABLE_READ は、読み取り固定です。
- SQL_TXN_SERIALIZABLE は、反復可能読み取りです。

SQL_DESCRIBE_PARAMETER (ストリング)
パラメーターが記述可能であれば "Y"、そうでなければ "N"。

SQL_DM_VER (ストリング)
予約済み。

SQL_DRIVER_BLDLEVEL
現行バージョンの CLI についてのレベル情報を構築します。

情報は s*YYMMDD* という以下のフォーマットになります。ここで、*YY* は構築の年、*MM* は月、*DD* は日です。例えば、**s100610** です。

特殊な構築の場合、フォーマットは、special_*JOBID* です (*JOBID* は特殊な構築のジョブ ID)。例えば、**special_39899** です。

完全なバージョン情報は、SQL_DRIVER_BLDLEVEL を SQL_DRIVER_VER と共に使用します。

SQL_DRIVER_HDBC (32 ビット)
CLI のデータベース・ハンドル

SQL_DRIVER_HDESC (32 ビット)
CLI の記述子ハンドル

SQL_DRIVER_HENV (32 ビット)
CLI の環境ハンドル

SQL_DRIVER_HLIB (32 ビット)
予約済み。

SQL_DRIVER_HSTMT (32 ビット)
CLI のステートメント・ハンドル

ODBC Driver Manager のある ODBC 環境では、*InfoType* を SQL_DRIVER_HSTMT に設定すると、ドライバー・マネージャーのステートメント・ハンドル (SQLAllocStmt() から返されるもの) がアプリケーションから *rgbInfoValue* の入力に渡されなければなりません。この場合、*rgbInfoValue* は入力引数と出力引数の両方です。 ODBC Driver Manager は、マップされた値を返します。 ODBC アプリケーションが CLI 固有関数 (LOB 関数など) を呼び出す場合、そのアプリケーションは、CLI ライブラリーをロードし、オペレーティング・システム関数呼び出しを発行して必要な関数を呼び出した後でハンドル値をその関数に渡すことにより、その関数にアクセスできます。

SQL_DRIVER_NAME (ストリング)
CLI インプリメンテーションのファイル名。

SQL_DRIVER_ODBC_VER (ストリング)
CLIがサポートする ODBC のバージョン番号。デフォルトでは、CLI は 『03.51』 を返します。 SQLSetEnvAttr() 関数を呼び出して、ODBC ドライバーのバージョンを変更できます。SQL_ATTR_ODBC_VERSION 属性を SQL_OV_ODBC3_80 (値 380) に設定すると、CLI は 『03.80』 を返します。

SQL_DRIVER_VER (ストリング)
IBM Data Server Driver for ODBC and CLI のバージョン。書式 'mm.vv.rrrr' のストリング (*mm* は主バージョン、*vv* は副バージョン、および *rrrr* はリリース)。例えば、"05.01.0000" は主バージョン 5、副バージョン 1、リリース 0 に変換します。完全なバージョン情報については、SQL_DRIVER_VER と共に SQL_DRIVER_BLDLEVEL を使用します。

SQL_DROP_ASSERTION (32 ビット符号なし整数)
DROP ASSERTION ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP ASSERTION ステートメントはサポートしません。

ODBC はさらに、CLI が返さない SQL_DA_DROP_ASSERTION 値を定義します。

SQL_DROP_CHARACTER_SET (32 ビット符号なし整数)
DROP CHARACTER SET ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP CHARACTER SET ステートメントはサポートしません。

ODBC はさらに、CLI が返さない SQL_DCS_DROP_CHARACTER_SET 値を定義します。

SQL_DROP_COLLATION (32 ビット符号なし整数)
DROP COLLATION ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP COLLATION ステートメントはサポートしません。

ODBC はさらに、CLI が返さない SQL_DC_DROP_COLLATION 値を定義します。

SQL_DROP_DOMAIN (32 ビット符号なし整数)
DROP DOMAIN ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP DOMAIN ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_DD_DROP_DOMAIN
- SQL_DD_CASCADE
- SQL_DD_RESTRICT

SQL_DROP_MODULE (32 ビット符号なし整数)
DROP MODULE ステートメント中のどの節を DBMS がサポートしているかを示します。 DB2 for z/OS の場合、CLI は常にゼロを戻します。

CLI は、次の値を戻します。
- SQL_DM_DROP_MODULE
- SQL_DM_RESTRICT

SQL_DROP_SCHEMA (32 ビット符号なし整数)
DROP SCHEMA ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP SCHEMA ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_DS_CASCADE
- SQL_DS_RESTRICT

SQL_DROP_TABLE (32 ビット符号なし整数)
DROP TABLE ステートメント中のどの節を DBMS がサポートしているかを示します。有効な戻り値は次のとおりです。
- SQL_DT_DROP_TABLE
- SQL_DT_CASCADE
- SQL_DT_RESTRICT

SQL_DROP_TRANSLATION (32 ビット符号なし整数)
DROP TRANSLATION ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP TRANSLATION ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。

- SQL_DTR_DROP_TRANSLATION

SQL_DROP_VIEW (32 ビット符号なし整数)
DROP VIEW ステートメント中のどの節を DBMS がサポートしているかを示します。 CLI は常にゼロを返し、DROP VIEW ステートメントはサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_DV_CASCADE
- SQL_DV_RESTRICT

SQL_DTC_TRANSITION_COST (32 ビット符号なしマスク)
接続への参加プロセスが高価かどうか判別する際に、 Microsoft Transaction Server が使用します。CLI は、以下の値を返します。
- SQL_DTC_ENLIST_EXPENSIVE
- SQL_DTC_UNENLIST_EXPENSIVE

SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (32 ビット・マスク)
CLI がサポートする動的カーソルの属性を示します (サブセット 1/2)。有効な戻り値は次のとおりです。
- SQL_CA1_NEXT
- SQL_CA1_ABSOLUTE
- SQL_CA1_RELATIVE
- SQL_CA1_BOOKMARK
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (32 ビット・マスク)
CLI がサポートする動的カーソルの属性を示します (サブセット 2/2)。有効な戻り値は次のとおりです。
- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_OPT_ROWVER_CONCURRENCY
- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_INSERT

- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_EXPRESSIONS_IN_ORDERBY (ストリング)
文字ストリング "Y" は、データベース・サーバーが ORDER BY リストの式の DIRECT 仕様をサポートしていることを示し、"N" は、そのサポートがないことを示します。

SQL_FETCH_DIRECTION (32 ビット・マスク)
サポートされているフェッチ方向。

以下のビット・マスクとフラグを使用して、どのオプションがサポートされているかを判別します。
- SQL_FD_FETCH_NEXT
- SQL_FD_FETCH_FIRST
- SQL_FD_FETCH_LAST
- SQL_FD_FETCH_PREV
- SQL_FD_FETCH_ABSOLUTE
- SQL_FD_FETCH_RELATIVE
- SQL_FD_FETCH_RESUME

SQL_FILE_USAGE (16 ビット整数)
単一階層のドライバーが、データ・ソース内のファイルをどのように直接扱うかを示します。 IBM Data Server Driver for ODBC and CLI ドライバーは単一層ドライバーではないため、常に SQL_FILE_NOT_SUPPORTED を返します。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_FILE_TABLE
- SQL_FILE_CATALOG

SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1 (32 ビット・マスク)
CLI がサポートする前方スクロール・カーソルの属性を示します。有効な戻り値は次のとおりです (サブセット 1/2)。
- SQL_CA1_NEXT
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH

- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2 (32 ビット・マスク)
CLI がサポートする前方スクロール・カーソルの属性を示します。有効な戻り値は次のとおりです (サブセット 2/2)。

- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_OPT_ROWVER_CONCURRENCY
- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_GETDATA_EXTENSIONS (32 ビット・マスク)
SQLGetData() 関数に対する拡張がサポートされているかどうかを示します。CLI では、現在以下の拡張が識別されサポートされています。

- SQL_GD_ANY_COLUMN。最後のバインド済み列より前にあるバインドされていない列について SQLGetData() を呼び出せます。
- SQL_GD_ANY_ORDER。どの順序の列についても SQLGetData() を呼び出せます。

ODBC はさらに、CLI が返さない以下の拡張機能を定義します。

- SQL_GD_BLOCK
- SQL_GD_BOUND

SQL_GROUP_BY (16 ビット整数)
サーバーによる、GROUP BY 節のサポートの度合いを示します。有効な戻り値は次のとおりです。

- SQL_GB_NO_RELATION - GROUP BY 節の列と SELECT リストの列の間のリレーションシップはありません。
- SQL_GB_NOT_SUPPORTED - GROUP BY 節はサポートされていません。
- SQL_GB_GROUP_BY_EQUALS_SELECT - GROUP BY 節には、SELECT リスト中のすべての非集約列が含まれていなければなりません。
- SQL_GB_GROUP_BY_CONTAINS_SELECT - GROUP BY 節には、SELECT リスト中のすべての非集約列が含まれていなければなりません。

- SQL_GB_COLLATE - COLLATE 節を各グループ化列の最後に指定できます。

SQL_IDENTIFIER_CASE (16 ビット整数)
オブジェクト名 (例えば、表名) の大/小文字の区別を示します。

有効な戻り値は次のとおりです。
- SQL_IC_UPPER : 大文字で保管されます。
- SQL_IC_LOWER : 小文字で保管されます。
- SQL_IC_SENSITIVE : 大文字小文字の区別があり、混合文字で保管されます。
- SQL_IC_MIXED : 大文字小文字の区別がなく、混合文字で保管されます。

注: IBM DBMS の ID 名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)
区切り ID を囲むために使用する文字を示します。

SQL_INDEX_KEYWORDS (32 ビット・マスク)
CREATE INDEX ステートメントでサポートされるキーワードを示します。有効な戻り値は次のとおりです。
- SQL_IK_NONE - どのキーワードもサポートされません。
- SQL_IK_ASC - ASC キーワードがサポートされます。
- SQL_IK_DESC - DESC キーワードがサポートされます。
- SQL_IK_ALL - すべてのキーワードがサポートされます。

CREATE INDEX ステートメントがサポートされることを確認するため、アプリケーションは `SQLGetInfo()` 関数を、SQL_DLL_INDEX *InfoType* 引数を指定して呼び出すことができます。

SQL_INFO_SCHEMA_VIEWS (32 ビット・マスク)
INFORMATION_SCHEMA 内のサポートされるビューを示します。 CLI は常にゼロを返し、INFORMATION_SCHEMA 中のビューをサポートしません。

ODBC はさらに、CLI が返さない以下の値を定義します。
- SQL_ISV_ASSERTIONS
- SQL_ISV_CHARACTER_SETS
- SQL_ISV_CHECK_CONSTRAINTS
- SQL_ISV_COLLATIONS
- SQL_ISV_COLUMN_DOMAIN_USAGE
- SQL_ISV_COLUMN_PRIVILEGES
- SQL_ISV_COLUMNS
- SQL_ISV_CONSTRAINT_COLUMN_USAGE
- SQL_ISV_CONSTRAINT_TABLE_USAGE
- SQL_ISV_DOMAIN_CONSTRAINTS
- SQL_ISV_DOMAINS
- SQL_ISV_KEY_COLUMN_USAGE
- SQL_ISV_REFERENTIAL_CONSTRAINTS
- SQL_ISV_SCHEMATA
- SQL_ISV_SQL_LANGUAGES
- SQL_ISV_TABLE_CONSTRAINTS

- SQL_ISV_TABLE_PRIVILEGES
- SQL_ISV_TABLES
- SQL_ISV_TRANSLATIONS
- SQL_ISV_USAGE_PRIVILEGES
- SQL_ISV_VIEW_COLUMN_USAGE
- SQL_ISV_VIEW_TABLE_USAGE
- SQL_ISV_VIEWS

SQL_INSERT_STATEMENT (32 ビット・マスク)
INSERT ステートメントのサポートを示します。有効な戻り値は次のとおりです。
- SQL_IS_INSERT_LITERALS
- SQL_IS_INSERT_SEARCHED
- SQL_IS_SELECT_INTO

SQL_INTEGRITY (ストリング)
文字ストリング "Y" は SQL89 および X/Open XPG4 組み込み SQL でデータ・ソースが整合性拡張機能 (IEF) をサポートしていることを示し、"N" はそのサポートがないことを示します。

CLI の前のバージョンでは、この *InfoType* 引数は SQL_ODBC_SQL_OPT_IEF でした。

SQL_KEYSET_CURSOR_ATTRIBUTES1 (32 ビット・マスク)
CLI がサポートするキー・セット駆動カーソルの属性を示します。有効な戻り値は次のとおりです (サブセット 1/2)。
- SQL_CA1_NEXT
- SQL_CA1_ABSOLUTE
- SQL_CA1_RELATIVE
- SQL_CA1_BOOKMARK
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_KEYSET_CURSOR_ATTRIBUTES2 (32 ビット・マスク)
CLI がサポートするキー・セット駆動カーソルの属性を示します。有効な戻り値は次のとおりです (サブセット 2/2)。
- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_OPT_ROWVER_CONCURRENCY

- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_KEYWORDS (ストリング)
データ・ソース固有のすべてのキーワードのコンマで区切ったリストを示します。この文字ストリングは、予約されているすべてのキーワードのリストです。相互操作可能なアプリケーションは、オブジェクト名中でそのようなキーワードを使用してはなりません。このリストには、ODBC 独自のキーワードや、データ・ソースと ODBC の両方で使用されるキーワードは載っていません。

SQL_LIKE_ESCAPE_CLAUSE (ストリング)
LIKE 述部内のパーセント文字 (%) と下線 (_) 用のエスケープ文字をデータ・ソースがサポートするかどうかを示します。また、LIKE 述部のエスケープ文字の定義用の ODBC 構文をドライバーがサポートするかどうかも示します。

- "Y" は、LIKE 述部内のエスケープ文字のサポートがあることを示します。
- "N" は、LIKE 述部内のエスケープ文字のサポートがないことを示します。

SQL_LOCK_TYPES (32 ビット・マスク)
予約済みオプション。ビット・マスクにはゼロが返されます。

SQL_MAX_ASYNC_CONCURRENT_STATEMENTS (32 ビット符号なし整数)
CLI が特定の接続でサポートできる、非同期モードにあるアクティブな並行ステートメントの最大数。制限値が指定されていないか不明である場合、この値はゼロです。

SQL_MAX_BINARY_LITERAL_LEN (32 ビット符号なし整数)
SQL ステートメント内のバイナリー・リテラルの最大長 (SQLGetTypeInfo() で戻されるリテラルの接頭部と接尾部を除いた 16 進文字数) を指定する 32 ビットの符号なし整数値。例えばバイナリー・リテラル 0xFFAA は 4 の長さをもちます。最大長がない場合や不明である場合、この値はゼロに設定されます。

SQL_MAX_CATALOG_NAME_LEN (16 ビット整数)
データ・ソース内のカタログ名の最大長。最大長がないか不明である場合、この値はゼロです。

CLI の前のバージョンでは、この *fInfoType* 引数は SQL_MAX_QUALIFIER_NAME_LEN でした。

SQL_MAX_CHAR_LITERAL_LEN (32 ビット符号なし整数)
SQL ステートメントの文字リテラルの最大長 (バイト単位)。限度がない場合はゼロです。

SQL_MAX_COLUMN_NAME_LEN (16 ビット整数)
列名の最大長 (バイト単位)。限度がない場合はゼロです。

SQL_MAX_COLUMNS_IN_GROUP_BY (16 ビット整数)
サーバーが GROUP BY 節でサポートする列の最大数を示します。限度がない場合はゼロです。

SQL_MAX_COLUMNS_IN_INDEX (16 ビット整数)
サーバーが索引でサポートする列の最大数を示します。限度がない場合はゼロです。

SQL_MAX_COLUMNS_IN_ORDER_BY (16 ビット整数)
サーバーが ORDER BY 節でサポートする列の最大数を示します。限度がない場合はゼロです。

SQL_MAX_COLUMNS_IN_SELECT (16 ビット整数)
サーバーが SELECT リストでサポートする列の最大数を示します。限度がない場合はゼロです。

SQL_MAX_COLUMNS_IN_TABLE (16 ビット整数)
サーバーが基本表でサポートする列の最大数を示します。限度がない場合はゼロです。

SQL_MAX_CONCURRENT_ACTIVITIES (16 ビット整数)
CLI がサポートできるアクティブ環境の最大数。制限値が指定されていないか不明である場合、この値はゼロに設定されます。

CLI の前のバージョンでは、この *InfoType* 引数は SQL_ACTIVE_ENVIRONMENTS でした。

SQL_MAX_CURSOR_NAME_LEN (16 ビット整数)
カーソル名の最大長 (バイト単位)。最大長がないか不明である場合、この値はゼロです。

SQL_MAX_DRIVER_CONNECTIONS (16 ビット整数)
アプリケーションごとにサポートされているアクティブ接続の最大数。

限度がシステム・リソースによって異なる場合は、ゼロが返されます。

CLI の前のバージョンでは、この *InfoType* 引数は SQL_ACTIVE_CONNECTIONS でした。

SQL_MAX_IDENTIFIER_LEN (16 ビット整数)
ユーザー定義名に対してデータ・ソースがサポートする最大サイズ (文字単位)。

SQL_MAX_INDEX_SIZE (32 ビット符号なし整数)
サーバーが索引中の結合列のためにサポートする最大サイズをバイト単位で示します。限度がない場合はゼロです。

SQL_MAX_MODULE_NAME_LEN (16 ビット整数)
モジュール修飾子名の最大長をバイト単位で示します。

SQL_MAX_PROCEDURE_NAME_LEN (16 ビット整数)
プロシージャー名の最大長 (バイト単位)。

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の行でサポートする最大長をバイト単位で指定します。限度がない場合はゼロです。

SQL_MAX_ROW_SIZE_INCLUDES_LONG (ストリング)
SQL_MAX_ROW_SIZE *InfoType* 引数によって返される値に製品固有の長ストリング・データ・タイプが組み込まれることを示す場合は、"Y" に設定します。それ以外は、"N" に設定します。

SQL_MAX_SCHEMA_NAME_LEN (16 ビット整数)
スキーマ修飾子名の最大長 (バイト単位)。

CLI の前のバージョンでは、この *fInfoType* 引数は SQL_MAX_OWNER_NAME_LEN でした。

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をバイト単位で示します (ステートメント内の空白の数を含む)。

SQL_MAX_TABLE_NAME_LEN (16 ビット整数)
表名の最大長 (バイト単位)。

SQL_MAX_TABLES_IN_SELECT (16 ビット整数)
<query specification> 中の FROM 節の表名の最大数を示します。

SQL_MAX_USER_NAME_LEN (16 ビット整数)
<user identifier> の最大サイズを示します (バイト単位)。

SQL_MODULE_USAGE (32 ビット・マスク)
SQL ステートメントの実行時に、関連したモジュールのあるステートメントのタイプを示します。 DB2 for z/OS の場合、CLI は常にゼロを戻します。

SQL_MU_PROCEDURE_INVOCATION は、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ています。

SQL_MULT_RESULT_SETS (ストリング)
文字ストリング "Y" は、データベースが複数の結果セットをサポートしていることを示し、 "N" はそれをサポートしていないことを示します。

SQL_MULTIPLE_ACTIVE_TXN (ストリング)
複数の接続でアクティブなトランザクションが許可されるかどうかを示します。

- "Y" は、複数の接続でアクティブなトランザクションを持てることを示します。
- "N" は、アクティブなトランザクションは一度に 1 つの接続についてのみ可能であることを示します。 CLI は、整合分散作業単位 (CONNECT

TYPE 2) 接続の場合は "N" を返し (トランザクションまたは作業単位がすべての接続にわたるため)、他のすべての接続の場合は "Y" を返します。

SQL_NEED_LONG_DATA_LEN (ストリング)
ODBC を使用するために予約されている文字ストリングを示します。常に、『N』 が戻されます。

SQL_NON_NULLABLE_COLUMNS (16 ビット整数)
NULL 不可列がサポートされているかどうかを示します。有効な戻り値は次のとおりです。
- SQL_NNC_NON_NULL - NOT NULL として定義できます。
- SQL_NNC_NULL - NOT NULL として定義できません。

SQL_NULL_COLLATION (16 ビット整数)
結果セット内で NULL がソートされるかどうかを示します。有効な戻り値は次のとおりです。
- SQL_NC_HIGH - NULL 値は高位にソートされます。
- SQL_NC_LOW - NULL 値は低位にソートされます。

SQL_NUMERIC_FUNCTIONS (32 ビット・マスク)
ODBC スカラー数字関数がサポートされることを示します。これらの関数は ODBC ベンダー・エスケープ・シーケンスとともに使用することを目的としています。

以下のビット・マスクを用いて、どの数字関数がサポートされているかを確かめます。
- 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)
ODBC 適合性のレベルを示します。有効な戻り値は次のとおりです。
- SQL_OAC_NONE
- SQL_OAC_LEVEL1
- SQL_OAC_LEVEL2

SQL_ODBC_INTERFACE_CONFORMANCE (32 ビット符号なし整数)
CLI が準拠している ODBC 3.0 インターフェースのレベルを示します。
- SQL_OIC_CORE : すべての ODBC ドライバーが準拠していると期待される最小レベル。このレベルには、接続機能などの基本インターフェース・エレメント、 SQL ステートメントの作成と実行のための機能、基本的な結果セット・メタデータ機能、基本的なカタログ機能などが含まれます。
- SQL_OIC_LEVEL1 : 上記のような中核となる規格に準拠するレベルの機能に加え、両方向スクロール・カーソル、更新部分や削除部分の位置を示すブックマークなどを含むレベル。
- SQL_OIC_LEVEL2 : レベル 1 の規格に準拠するレベルの機能に加え、機密カーソル、ブックマークによる更新・削除・最新表示、ストアード・プロシージャーのサポート、主キーおよび外部キーに対するカタログ機能などの拡張フィーチャーが含まれているレベル。

SQL_ODBC_SAG_CLI_CONFORMANCE (16 ビット整数)
SQL アクセス・グループ (SAG) CLI 仕様の関数の適合性。

有効な戻り値は次のとおりです。
- SQL_OSCC_NOT_COMPLIANT : ドライバーは SAG に適合しません。
- SQL_OSCC_COMPLIANT : ドライバーは SAG に適合します。

SQL_ODBC_SQL_CONFORMANCE (16 ビット整数)
有効な戻り値は次のとおりです。
- SQL_OSC_MINIMUM : 最低限の ODBC SQL 文法がサポートされます。
- SQL_OSC_CORE : コア ODBC SQL 文法がサポートされます。
- SQL_OSC_EXTENDED : 拡張された ODBC SQL 文法がサポートされます。

SQL_ODBC_VER (ストリング)
ドライバー・マネージャーがサポートする ODBC のバージョン番号。

CLI は、ストリング 『03.01.0000』 を返します。CLI はストリング "03.01.0000" を返します。Windows 7 および Windows Server 2008 R2 オペレーティング・システムでは、CLI はストリング 『03.80.0000』 を返します。

SQL_OJ_CAPABILITIES (32 ビット・マスク)
サポートされている外部結合タイプを列挙する 32 ビット・マスク。

ビット・マスクは以下のとおりです。
- SQL_OJ_LEFT : 左方外部結合がサポートされています。
- SQL_OJ_RIGHT : 右方外部結合がサポートされています。
- SQL_OJ_FULL : 全外部結合がサポートされています。
- SQL_OJ_NESTED : ネスト外部結合がサポートされています。
- SQL_OJ_ORDERED : 外部結合 ON 節の列の基礎となる表の順序は、JOIN 節の表の順序と同じである必要はありません。

- SQL_OJ_INNER : 外部結合の内部表を内部結合にすることもできます。
- SQL_OJ_ALL_COMPARISONS_OPS : 外部結合 ON 節内でどの述部でも使用できます。このビットを設定しない場合、外部結合内で使える比較演算子は等価演算子 (=) だけです。

SQL_ORDER_BY_COLUMNS_IN_SELECT (ストリング)
ORDER BY 節の列を選択リストに入れる必要がある場合は "Y" に設定してください。必要がない場合は "N" に設定してください。

SQL_OUTER_JOINS (ストリング)
以下の文字ストリングは、以下の意味で使用されています。
- "Y" は、外部結合がサポートされていて、 CLI 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、外部結合がサポートされていないことを示します。

SQL_PARAM_ARRAY_ROW_COUNTS (32 ビット符号なし整数)
パラメーター化実行における行カウントの可用性を示します。
- SQL_PARC_BATCH : 個々の行カウントをパラメーターの各セットで使用できます。この動作は概念的には、SQL ステートメントのバッチを生成する CLI と同等であり、配列内の各パラメーター・セットについて 1 つあります。 SQL_PARAM_STATUS_PTR 記述子フィールドを利用すれば、拡張エラー情報を検索できます。非アトミック操作でこの動作を使用可能にするには、SQL_ATTR_PARC_BATCH 接続属性を SQL_PARC_BATCH_ENABLE に設定し、SQL_ATTR_PARAMOPT_ATOMIC を SQL_ATOMIC_NO に設定します。SQL_ATTR_PARAMOPT_ATOMIC が SQL_ATOMIC_YES に設定されていると、CLI0150E エラー・メッセージが返されます。
- SQL_PARC_NO_BATCH : 使用できる行カウントは 1 つしかなく、それはパラメーターの配列全体に対するステートメントの実行により生じる累積行カウントです。この動作は概念的には、ステートメントとパラメーター配列全体とを 1 つのアトミック単位として扱うのと同等です。エラーの処理は、1 つのステートメントを実行する場合と同じように行われます。

SQL_PARAM_ARRAY_SELECTS (32 ビット符号なし整数)
パラメーター化実行における結果セットの可用性を示します。有効な戻り値は次のとおりです。
- SQL_PAS_BATCH : パラメーターのセットにつき使用できる結果セットは 1 つです。SQL_PAS_BATCH は概念的には、SQL ステートメントのバッチを生成する CLI と同等であり、配列内のパラメーター・セットごとに 1 つずつあります。
- SQL_PAS_NO_BATCH : 使用できる結果セットは 1 つしかなく、それはパラメーターの配列全体に対するステートメントの実行により生じる累積結果セットを表します。SQL_PAS_NO_BATCH は概念的には、ステートメントとパラメーター配列全体とを 1 つのアトミック単位として扱うのと同等です。
- SQL_PAS_NO_SELECT : 結果セットを生成するステートメントにパラメーターの配列を指定して実行することを、CLI が許可しません。

SQL_POS_OPERATIONS (32 ビット・マスク)
予約済みオプション。ビット・マスクにはゼロが返されます。

SQL_POSITIONED_STATEMENTS (32 ビット・マスク)
定位置 UPDATE および定位置 DELETE ステートメントがサポートされている程度を示します。
- SQL_PS_POSITIONED_DELETE
- SQL_PS_POSITIONED_UPDATE
- SQL_PS_SELECT_FOR_UPDATE - これは、カーソルを使用して列を更新できるようにするために、サーバーが FOR UPDATE 節を <query expression> に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)
データベース・ベンダーがプロシージャー用に使用している名前。

SQL_PROCEDURES (ストリング)
文字ストリング "Y" は、データ・ソースがプロシージャーをサポートしていて、しかも CALL ステートメントで指定された ODBC プロシージャー呼び出し構文を CLI がサポートしていることを示します。 "N" は、サポートされていないことを示します。

SQL_QUOTED_IDENTIFIER_CASE (16 ビット整数)
有効な戻り値は次のとおりです。
- SQL_IC_UPPER : 大/小文字の区別がなく、大文字で保管されます。
- SQL_IC_LOWER : 大/小文字の区別がなく、小文字で保管されます。
- SQL_IC_SENSITIVE : SQL 中の引用符付き ID (区切り ID) は大文字小文字の区別があり、混合文字でシステム・カタログに保管されます。
- SQL_IC_MIXED - 大/小文字の区別がなく、大/小文字混合で保管されます。

SQL_QUOTED_ IDENTIFIER_CASE 整数を、(引用符なし) ID をシステム・カタログに保管する方法の判別に使用される SQL_IDENTIFIER_CASE *InfoType* 引数と対比させてください。

SQL_ROW_UPDATES (ストリング)
文字ストリング "Y" の場合、キー・セット・ドリブン・カーソルまたは混合カーソルが、フェッチされたすべての行のバージョンまたは値を維持するので、行の最後のフェッチ以後に行に対して加えられた更新を検出できることを意味します。この文字ストリングに該当するのは、削除や追加ではなく、更新だけです。`SQLFetchScroll()` を呼び出した場合、CLI は SQL_ROW_UPDATED フラグを行状況配列に戻すことがあります。それ以外の場合は、"N" が返されます。

SQL_SCHEMA_TERM (ストリング)
データベース・ベンダーのスキーマ (所有者) 用の用語。

CLI の前のバージョンでは、この *InfoType* は SQL_OWNER_TERM でした。

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。以下の有効なスキーマ修飾子 (所有者) が戻されます。
- SQL_SU_DML_STATEMENTS - すべての DML ステートメント。
- SQL_SU_PROCEDURE_INVOCATION - プロシージャー呼び出しステートメント。

- SQL_SU_TABLE_DEFINITION - すべての表定義ステートメント。
- SQL_SU_INDEX_DEFINITION - すべての索引定義ステートメント。
- SQL_SU_PRIVILEGE_DEFINITION - すべての特権定義ステートメント (GRANT ステートメントと REVOKE ステートメント)。

CLI の前のバージョンでは、この *InfoType* 引数は SQL_OWNER_USAGE でした。

SQL_SCROLL_CONCURRENCY (32 ビット・マスク)
カーソル用にサポートされている並行性オプションを示します。

以下のビット・マスクとフラグを使用して、どのオプションがサポートされているかを判別します。
- SQL_SCCO_LOCK
- SQL_SCCO_READ_ONLY
- SQL_SCCO_TIMESTAMP
- SQL_SCCO_VALUES

CLI は SQL_SCCO_LOCK を戻し、更新を行うために十分なロックのうち最低のレベルを示します。

SQL_SCROLL_OPTIONS (32 ビット・マスク)
両方向スクロール・カーソルをサポートするスクロール・オプションを示します。

以下のビット・マスクとフラグを使用して、どのオプションがサポートされているかを判別します。
- SQL_SO_FORWARD_ONLY: カーソルは前方スクロールのみ可能です。
- SQL_SO_KEYSET_DRIVEN: CLI は、結果セット内のすべての行のキーを保管して使用します。
- SQL_SO_STATIC: 結果セット内のデータは、静的データです。
- SQL_SO_DYNAMIC: CLI は、行セット内の各行のキーを保存します (キー・セットのサイズは行セットのサイズと同じです)。
- SQL_SO_MIXED: CLI は、キー・セット内の各行のキーを保存しますが、キー・セットのサイズは行セットのサイズより大きいです。カーソルは、キー・セット内部ではキー・セット主導型ですが、キー・セット外部では動的カーソルです。

SQL_SEARCH_PATTERN_ESCAPE (ストリング)
SQLTables() 関数や SQLColumns() 関数などのカタログ関数用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します。

SQL_SERVER_NAME (ストリング)
DB2 インスタンスの名前を示します。この文字ストリングは SQL_DATA_SOURCE_NAME 文字ストリングとは対照的に、データベース・サーバーの実際の名前です。DBMS の中には、データベースの実際のサーバー名と違う名前を提供して接続を確立するものもあります。

SQL_SPECIAL_CHARACTERS (ストリング)
データ・ソースにおいて表、列、索引の名前などの ID 名内で使用できる特殊文字 (a...z、A...Z、0...9、および下線以外のすべての文字) のみを使用した文字ストリング。例えば "@#" などがあります。ID 内で特殊文字を使用する場合、その ID は区切り ID でなければなりません。

SQL_SQL_CONFORMANCE (32 ビット符号なし整数)
サポートしている SQL-92 のレベルを示します。
- SQL_SC_SQL92_ENTRY : 項目レベルの SQL-92 に準拠。
- SQL_SC_FIPS127_2_TRANSITIONAL : FIPS 127-2 遷移レベルに準拠。
- SQL_SC_SQL92_FULL : 完全レベルの SQL-92 に準拠。
- SQL_SC_ SQL92_INTERMEDIATE : 中間レベルの SQL-92 に準拠。

SQL_SQL92_DATETIME_FUNCTIONS (32 ビット・マスク)
CLI およびデータ・ソースがサポートする日時スカラー関数を示します。有効な戻り値は次のとおりです。
- SQL_SDF_CURRENT_DATE
- SQL_SDF_CURRENT_TIME
- SQL_SDF_CURRENT_TIMESTAMP

SQL_SQL92_FOREIGN_KEY_DELETE_RULE (32 ビット・マスク)
DELETE ステートメント中の外部キーに対してサポートされる規則 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SFKD_CASCADE
- SQL_SFKD_NO_ACTION
- SQL_SFKD_SET_DEFAULT
- SQL_SFKD_SET_NULL

SQL_SQL92_FOREIGN_KEY_UPDATE_RULE (32 ビット・マスク)
UPDATE ステートメント中の外部キーに対してサポートされる規則 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SFKU_CASCADE
- SQL_SFKU_NO_ACTION
- SQL_SFKU_SET_DEFAULT
- SQL_SFKU_SET_NULL

SQL_SQL92_GRANT (32 ビット・マスク)
GRANT ステートメント中でサポートされる節 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SG_DELETE_TABLE
- SQL_SG_INSERT_COLUMN
- SQL_SG_INSERT_TABLE
- SQL_SG_REFERENCES_TABLE
- SQL_SG_REFERENCES_COLUMN
- SQL_SG_SELECT_TABLE
- SQL_SG_UPDATE_COLUMN
- SQL_SG_UPDATE_TABLE
- SQL_SG_USAGE_ON_DOMAIN
- SQL_SG_USAGE_ON_CHARACTER_SET
- SQL_SG_USAGE_ON_COLLATION
- SQL_SG_USAGE_ON_TRANSLATION
- SQL_SG_WITH_GRANT_OPTION

SQL_SQL92_NUMERIC_VALUE_FUNCTIONS (32 ビット・マスク)
CLI およびデータ・ソースがサポートする数値スカラー関数 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SNVF_BIT_LENGTH

- SQL_SNVF_CHAR_LENGTH
- SQL_SNVF_CHARACTER_LENGTH
- SQL_SNVF_EXTRACT
- SQL_SNVF_OCTET_LENGTH
- SQL_SNVF_POSITION

SQL_SQL92_PREDICATES (32 ビット・マスク)
SELECT ステートメント中でサポートされる述部 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。

- 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_RELATIONAL_JOIN_OPERATORS (32 ビット・マスク)
SELECT ステートメント中でサポートされる関係結合演算子 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。

- SQL_SRJO_CORRESPONDING_CLAUSE
- SQL_SRJO_CROSS_JOIN
- SQL_SRJO_EXCEPT_JOIN
- SQL_SRJO_FULL_OUTER_JOIN
- SQL_SRJO_INNER_JOIN (内部結合機能ではなく、INNER JOIN 構文のサポートを示します)
- SQL_SRJO_INTERSECT_JOIN
- SQL_SRJO_LEFT_OUTER_JOIN
- SQL_SRJO_NATURAL_JOIN
- SQL_SRJO_RIGHT_OUTER_JOIN
- SQL_SRJO_UNION_JOIN

SQL_SQL92_REVOKE (32 ビット・マスク)
REVOKE ステートメント中でデータ・ソースがサポートする節 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。

- SQL_SR_CASCADE
- SQL_SR_DELETE_TABLE
- SQL_SR_GRANT_OPTION_FOR
- SQL_SR_INSERT_COLUMN
- SQL_SR_INSERT_TABLE
- SQL_SR_REFERENCES_COLUMN
- SQL_SR_REFERENCES_TABLE
- SQL_SR_RESTRICT

- SQL_SR_SELECT_TABLE
- SQL_SR_UPDATE_COLUMN
- SQL_SR_UPDATE_TABLE
- SQL_SR_USAGE_ON_DOMAIN
- SQL_SR_USAGE_ON_CHARACTER_SET
- SQL_SR_USAGE_ON_COLLATION
- SQL_SR_USAGE_ON_TRANSLATION

SQL_SQL92_ROW_VALUE_CONSTRUCTOR (32 ビット・マスク)
SELECT ステートメント中でサポートされる行値コンストラクター式 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SRVC_DEFAULT
- SQL_SRVC_NULL
- SQL_SRVC_ROW_SUBQUERY
- SQL_SRVC_VALUE_EXPRESSION

SQL_SQL92_STRING_FUNCTIONS (32 ビット・マスク)
CLI およびデータ・ソースがサポートするストリング・スカラー関数 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SSF_CONVERT
- SQL_SSF_LOWER
- SQL_SSF_SUBSTRING
- SQL_SSF_TRANSLATE
- SQL_SSF_TRIM_BOTH
- SQL_SSF_TRIM_LEADING
- SQL_SSF_TRIM_TRAILING
- SQL_SSF_UPPER

SQL_SQL92_VALUE_EXPRESSIONS (32 ビット・マスク)
サポートされる値式 (SQL-92 により定義) を示します。有効な戻り値は次のとおりです。
- SQL_SVE_CASE
- SQL_SVE_CAST
- SQL_SVE_COALESCE
- SQL_SVE_NULLIF

SQL_STANDARD_CLI_CONFORMANCE (32 ビット・マスク)
CLI が準拠している 1 つ以上の CLI 標準を示します。有効な戻り値は次のとおりです。
- SQL_SCC_ISO92_CLI
- SQL_SCC_XOPEN_CLI_VERSION1

SQL_STATIC_CURSOR_ATTRIBUTES1 (32 ビット・マスク)
CLI がサポートする静的カーソルの属性を示します。有効な戻り値は次のとおりです (サブセット 1/2)。
- SQL_CA1_ABSOLUTE
- SQL_CA1_BOOKMARK
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK

- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_NEXT
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_REFRESH
- SQL_CA1_POS_UPDATE
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_RELATIVE
- SQL_CA1_SELECT_FOR_UPDATE

SQL_STATIC_CURSOR_ATTRIBUTES2 (32 ビット・マスク)
CLI がサポートする静的カーソルの属性を示します (サブセット 2/2)。

- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_OPT_ROWVER_CONCURRENCY
- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_STATIC_SENSITIVITY (32 ビット・マスク)
アプリケーションが定位置更新や削除によって加えた変更を、そのアプリケーションが検出できるかどうかを示します。有効な戻り値は次のとおりです。

- SQL_SS_ADDITIONS : カーソルは追加された行を認識でき、カーソルは追加された行へスクロールできます。すべての DB2 サーバーは追加された行を認識できます。
- SQL_SS_DELETIONS : カーソルは削除された行を使用できなくなり、結果セットに空いた穴を残しません。削除された行からカーソルがスクロールした後、その行に戻ることはできません。
- SQL_SS_UPDATES : カーソルは、更新された行を認識できます。カーソルが、更新された行からスクロールした後にその行に戻ると、そのカーソルから返されるデータは更新されたデータになり、元のデータは返されません。

SQL_STRING_FUNCTIONS (32 ビット・マスク)
どのストリング関数がサポートされているかを示します。

以下のビット・マスクを用いて、どのストリング関数がサポートされているかを示します。
- SQL_FN_STR_ASCII
- SQL_FN_STR_BIT_LENGTH
- SQL_FN_STR_CHAR
- SQL_FN_STR_CHAR_LENGTH
- SQL_FN_STR_CHARACTER_LENGTH
- SQL_FN_STR_CONCAT
- SQL_FN_STR_DIFFERENCE
- SQL_FN_STR_INSERT
- SQL_FN_STR_LCASE
- SQL_FN_STR_LEFT
- SQL_FN_STR_LENGTH
- SQL_FN_STR_LOCATE
- SQL_FN_STR_LOCATE_2
- SQL_FN_STR_LTRIM
- SQL_FN_STR_OCTET_LENGTH
- SQL_FN_STR_POSITION
- SQL_FN_STR_REPEAT
- 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、*string_exp1*、*string_exp2*、 および *start* 引数を指定した LOCATE スカラー関数を呼び出せる場合は、SQL_FN_STR_LOCATE ビット・マスクが返されます。アプリケーションが呼び出せるのが、*string_exp1* および *string_exp2* 引数を指定した LOCATE スカラー関数だけである場合は、SQL_FN_STR_LOCATE_2 ビット・マスクが返されます。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・マスクが返されます。

SQL_SUBQUERIES (32 ビット・マスク)
副照会をサポートしている述部を示します。有効な戻り値は次のとおりです。
- SQL_SQ_COMPARISION : *comparison* (比較) 述部。
- SQL_SQ_CORRELATE_SUBQUERIES : 副照会をサポートするすべての述部は、相関副照会もサポートします。
- SQL_SQ_EXISTS : *exists* (存在) 述部。
- SQL_SQ_IN : *in* (～にある) 述部。
- SQL_SQ_QUANTIFIED : 多値比較スカラー関数を含む述部。

SQL_SYSTEM_FUNCTIONS (32 ビット・マスク)
どのスカラー・システム関数がサポートされているかを示します。

以下のビット・マスクを用いて、どのスカラー・システム関数がサポートされているかを示します。
- SQL_FN_SYS_DBNAME
- SQL_FN_SYS_IFNULL
- SQL_FN_SYS_USERNAME

注: これらの関数は、 ODBC のエスケープ・シーケンスと一緒に使用するためのものです。

SQL_TABLE_TERM (ストリング)
表に関するデータベース・ベンダーの用語。

SQL_TIMEDATE_ADD_INTERVALS (32 ビット・マスク)
特殊な ODBC システム関数 TIMESTAMPADD がサポートされているかどうか、および、サポートされている場合はサポートされているインターバルを示します。

以下のビット・マスクを用いて、どのインターバルがサポートされているかを示します。
- SQL_FN_TSI_FRAC_SECOND
- SQL_FN_TSI_SECOND
- SQL_FN_TSI_MINUTE
- SQL_FN_TSI_HOUR
- SQL_FN_TSI_DAY
- SQL_FN_TSI_WEEK
- SQL_FN_TSI_MONTH
- SQL_FN_TSI_QUARTER
- SQL_FN_TSI_YEAR

SQL_TIMEDATE_DIFF_INTERVALS (32 ビット・マスク)
特殊な ODBC システム関数 TIMESTAMPDIFF がサポートされているかどうか、および、サポートされている場合はサポートされているインターバルを示します。

以下のビット・マスクを用いて、どのインターバルがサポートされているかを示します。
- SQL_FN_TSI_FRAC_SECOND
- SQL_FN_TSI_SECOND
- SQL_FN_TSI_MINUTE
- SQL_FN_TSI_HOUR
- SQL_FN_TSI_DAY
- SQL_FN_TSI_WEEK
- SQL_FN_TSI_MONTH
- SQL_FN_TSI_QUARTER
- SQL_FN_TSI_YEAR

SQL_TIMEDATE_FUNCTIONS (32 ビット・マスク)
どの日時関数がサポートされているかを示します。

以下のビット・マスクを用いて、どの日時関数がサポートされているかを示します。
- SQL_FN_TD_CURRENT_DATE
- SQL_FN_TD_CURRENT_TIME

- SQL_FN_TD_CURRENT_TIMESTAMP
- SQL_FN_TD_CURDATE
- SQL_FN_TD_CURTIME
- SQL_FN_TD_DAYNAME
- SQL_FN_TD_DAYOFMONTH
- SQL_FN_TD_DAYOFWEEK
- SQL_FN_TD_DAYOFYEAR
- SQL_FN_TD_EXTRACT
- SQL_FN_TD_HOUR
- SQL_FN_TD_JULIAN_DAY
- SQL_FN_TD_MINUTE
- SQL_FN_TD_MONTH
- SQL_FN_TD_MONTHNAME
- SQL_FN_TD_NOW
- SQL_FN_TD_QUARTER
- SQL_FN_TD_SECOND
- SQL_FN_TD_SECONDS_SINCE_MIDNIGHT
- SQL_FN_TD_TIMESTAMPADD
- SQL_FN_TD_TIMESTAMPDIFF
- SQL_FN_TD_WEEK
- SQL_FN_TD_YEAR

注: これらの関数は、 ODBC のエスケープ・シーケンスと一緒に使用するためのものです。

SQL_TXN_CAPABLE (16 ビット整数)

トランザクションに DDL、DML、またはその両方が含まれているかどうかを示します。有効な戻り値は次のとおりです。

- 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 : トランザクションは DML ステートメント (例えば SELECT、INSERT、UPDATE、および DELETE) だけを含むことができます。トランザクション内に CREATE TABLE や DROP INDEX などの DDL ステートメントがあると、エラーの原因となります。
- SQL_TC_DDL_COMMIT : トランザクションは DML ステートメントだけを含むことができます。トランザクションに DDL ステートメントがあると、そのトランザクションはコミットされます。
- SQL_TC_DDL_IGNORE : トランザクションは DML ステートメントだけを含むことができます。トランザクションに DDL ステートメントがあると、そのステートメントは無視されます。
- SQL_TC_ALL : トランザクションは、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むことができます。

SQL_TXN_ISOLATION_OPTION (32 ビット・マスク)

現在接続中のデータベース・サーバーで使用できるトランザクション分離レベル。

以下のマスクとフラグを用いて、どのオプションがサポートされているかを示します。

- SQL_TXN_READ_UNCOMMITTED
- SQL_TXN_READ_COMMITTED

- SQL_TXN_REPEATABLE_READ
- SQL_TXN_SERIALIZABLE
- SQL_TXN_NOCOMMIT
- SQL_TXN_VERSIONING

個々のレベルに関する説明は、SQL_DEFAULT_TXN_ISOLATION を参照してください。

SQL_UNION (32 ビット・マスク)
サーバーが UNION 演算子をサポートしているかどうかを示します。有効な戻り値は次のとおりです。
- SQL_U_UNION : UNION 節をサポートしています。
- SQL_U_UNION_ALL : UNION 節の ALL キーワードをサポートしています。

SQL_U_UNION_ALL が設定されている場合は、SQL_U_UNION です。

SQL_USER_NAME (ストリング)
特定のデータベースで使用されるユーザー名を示します。この文字ストリングは SQLConnect() 呼び出しで指定される ID です。

SQL_XOPEN_CLI_YEAR (ストリング)
該当バージョンのドライバーが完全に準拠している X/Open 仕様の資料の年度を示します。

SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し

現行トランザクション中に (フェッチまたは SQLGetSubString() 呼び出しの結果として) サーバーから戻されたラージ・オブジェクト・ロケーターによって参照される、ラージ・オブジェクト値の長さを検索します。

仕様:

- **CLI** 2.1

構文

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

関数引数

表 95. SQLGetLength 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドルこれは、既に割り振り済みではあるが、現在は準備済み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・ハンドルです。

表 95. SQLGetLength 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*LocatorCType*	入力	C タイプのソース LOB ロケーター。これは、次のいずれかです。 • 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する必要があります。
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であり、ロケーター値ではない場合に、 *rgbValue* に戻される情報の長さ (バイト数)[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	常にゼロに設定されます。

注:

a DBCLOB データの場合はこれは文字数です。

使用法

SQLGetLength() は、 LOB ロケーターで表されるデータ値の長さを判別するときに使用します。これは、参照される LOB 値の一部またはすべてを取得するための適切な方法を選択できるように、 LOB の全長を判別するためにアプリケーションが使用します。長さは、サーバー・コード・ページを使用してデータベース・サーバーによって計算されるため、アプリケーション・コード・ページとサーバー・コード・ページが違う場合には、クライアントでのスペース所要量の計算は少し複雑になる可能性があります。アプリケーションは、必要に応じてコード・ページ拡張に対応可能でなければなりません。

Locator 引数には、任意の有効な LOB ロケーターを入れられます。ただし、そのロケーターは、FREE LOCATOR ステートメントを用いて明示的に解放されたり、またはロケーターの作成時にトランザクションが終了したために暗黙的に解放されたりしていないものとします。

このステートメント・ハンドルは、準備済みステートメントまたはカタログ関数呼び出しに関連付けられていてはなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 96. SQLGetLength SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	*LocatorCType* と *Locator* の組み合わせは無効です。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY003	プログラム・タイプが範囲外です。	*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	引数の値が無効です。	*StringLength* を指すポインターが NULL でした。
HY010	関数のシーケンス・エラーです。	指定した *StatementHandle* は、*割り振られて* いません。 実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HYC00	ドライバーが使用できません。	アプリケーションは現在、ラージ・オブジェクトをサポートしないデータ・ソースに接続しています。
0F001	LOB トークン変数は、現在何も値を表していません。	*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。

制限

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場合は、この関数は使用できません。関数タイプを SQL_API_SQLGETLENGTH に設定して SQLGetFunctions() を呼び出し、 *fExists* 出力引数を調べて、現行の接続でその関数がサポートされているかどうかを判別してください。

例

```
/* get the length of the whole CLOB data */
cliRC = SQLGetLength(hstmtLocUse,
                     SQL_C_CLOB_LOCATOR,
                     clobLoc,
                     &clobLen,
                     &ind);
```

SQLGetPosition 関数 (CLI) - ストリングの開始位置を戻す

LOB 値 (ソース) 内の、あるストリングの開始位置を戻すときに使用します。

ソース値は LOB ロケーターでなければならず、検索ストリングは LOB ロケーターまたはリテラル・ストリングとすることができます。

仕様:

- **CLI** 2.1

ソースおよび検索 LOB ロケーターは、現行トランザクション中にフェッチまたは SQLGetSubString() 呼び出しによってデータベースから戻された任意のものです。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetPositionW() です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

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

関数引数

表 97. SQLGetPosition 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドルこれは、既に割り振り済みではあるが、現在は準備済み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・ハンドルです。
SQLSMALLINT	*LocatorCType*	入力	C タイプのソース LOB ロケーター。これには、以下の種類があります。 • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	入力	*Locator* 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 であって、しかも *SearchLiteralLength* を 0 に設定する場合、検索ストリングに関連付けられた LOB ロケーターに *SearchLocator* を設定しなければなりません。そうしない場合、この引数は無視されます。

表 97. SQLGetPosition 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLCHAR *	*SearchLiteral*	入力	この引数は、検索ストリング・リテラルを収容するストレージ域を指します。 *SearchLiteralLength* が 0 であれば、このポインターは NULL でなければなりません。
SQLINTEGER	*SearchLiteralLength*	入力	*SearchLiteral* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数 (バイト単位)。[a] この引数値が 0 の場合は、引数 *SearchLocator* は有効です。
SQLUINTEGER	*FromPosition*	入力	BLOB や CLOB の場合、これはソース・ストリング内で検索が始まる最初のバイト位置で、 DBCLOB の場合、これは最初の文字です。最初のバイトまたは文字には、番号 1 が付けられます。
SQLUINTEGER *	*LocatedAt*	出力	BLOB と CLOB の場合、これはストリングが置かれたバイト位置で、ストリングが置かれていない場合には値ゼロが戻されます。 DBCLOB の場合、これは文字位置です。 ソース・ストリングの長さがゼロの場合は、値 1 が戻されます。
SQLINTEGER *	*IndicatorValue*	出力	常にゼロに設定されます。

注:

a これは、DBCLOB データの場合であってもバイト単位です。

使用法

SQLGetPosition() は、SQLGetSubString() と一緒に使用して、 LOB の任意の部分をランダムな方法で取得します。 SQLGetSubString() を使用するには、ストリング全体の中のサブストリングのロケーションを事前に知っておく必要があります。サブストリングの先頭を検索ストリングで検出できる場合、 SQLGetPosition() を使用して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* (使用する場合) 引数には、任意の有効な LOB ロケーターを入れられます。ただし、そのロケーターは、FREE LOCATOR ステートメントを用いて明示的に解放されたり、またはロケーターの作成時のトランザクションが終了したために暗黙的に解放されたりしていないものとします。

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、準備済みステートメントまたはカタログ関数呼び出しに関連付けられていてはなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 98. SQLGetPosition SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	*LocatorCType* と LOB ロケーター値の組み合わせは無効です。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	*LocatedAt* 引数を指すポインターが NULL でした。 *FromPosition* の引数値は、0 より大きい値ではありませんでした。 *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。
HY010	関数のシーケンス・エラーです。	指定した *StatementHandle* は、*割り振られて* いません。 実行時データ (`SQLParamData()`、`SQLPutData()`) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	*SearchLiteralLength* の値が 1 より小さく、かつ SQL_NTS ではありませんでした。 パターンの長さは、関連付けられている変数 SQL データ・タイプの最大データ長より長くなります。(DB2 for z/OS サーバーの場合、パターンの長さは、データ・タイプまたは *LocatorCType* に関係なく、最大 4000 バイトです。) LocatorCType が SQL_C_CLOB_LOCATOR の場合は、リテラル最大サイズは SQLCLOB の最大サイズです。 LocatorCType が SQL_C_BLOB_LOCATOR の場合は、リテラル最大サイズは SQLVARBINARY の最大サイズです。 LocatorCType が SQL_C_DBCLOB_LOCATOR の場合は、リテラル最大サイズは SQLVARGRAPHIC の最大サイズです。

表 98. SQLGetPosition SQLSTATE (続き)

SQLSTATE	説明	解説
HYC00	ドライバーが使用できません。	アプリケーションは現在、ラージ・オブジェクトをサポートしないデータ・ソースに接続しています。
0F001	LOB トークン変数は、現在何も値を表していません。	*Locator* または *SearchLocator* で指定した値は現在、 LOB ロケーターではありません。

制限

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場合は、この関数は使用できません。関数タイプを SQL_API_SQLGETPOSITION に設定して SQLGetFunctions() を呼び出し、 *fExists* 出力引数を調べて、現行の接続でその関数がサポートされているかどうかを判別してください。

SQLGetPosition() 関数は、WCHAR データではなくグラフィック・データを処理することを目的としています。そのため、この関数に渡されるデータは、ビッグ・エンディアンとみなす必要があります。

例

```
/* get the starting position of the CLOB piece of data */
cliRC = SQLGetPosition(hstmtLocUse,
                       SQL_C_CLOB_LOCATOR,
                       clobLoc,
                       0,
                       (SQLCHAR *)"Interests",
                       strlen("Interests"),
                       1,
                       &clobPiecePos,
                       &ind);
```

SQLGetSQLCA 関数 (CLI) - SQLCA データ構造の取得

SQLGetSQLCA() は使用すべきでない関数になりました。 SQLGetDiagField() および SQLGetDiagRec() を使用して、診断情報を探してください。

このバージョンの CLI では引き続き SQLGetSQLCA() をサポートしますが、CLI プログラムではこの関数の使用をやめ、最新の規格に従うようお勧めします。

SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得

ステートメント属性の現行設定値を戻します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLGetStmtAttrW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLGetStmtAttr  (SQLHSTMT         StatementHandle,
                             SQLINTEGER       Attribute,
                             SQLPOINTER       ValuePtr,
                             SQLINTEGER       BufferLength,
                             SQLINTEGER       *StringLengthPtr);
```

関数引数

表 99. SQLGetStmtAttr 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLINTEGER	*Attribute*	入力	取り出す属性。
SQLPOINTER	*ValuePtr*	出力	*Attribute* に指定されている属性値を戻すバッファーを指すポインター。
SQLINTEGER	*BufferLength*	入力	*Attribute* が ODBC 定義の属性で、 *ValuePtr* が文字ストリングかバイナリー・バッファーを指している場合、この引数の長さは **ValuePtr* の長さにする必要があります。 *Attribute* が ODBC 定義の属性で、 **ValuePtr* が整数の場合、 *BufferLength* は無視されます。 *Attribute* が CLI 属性の場合、アプリケーションは *BufferLength* 引数を設定して、その属性の性質を示します。 *BufferLength* には以下の値が入ります。 • **ValuePtr* が文字ストリングを指すポインターの場合、*BufferLength* は、そのストリングを格納するのに必要なバイト数であるか、または SQL_NTS です。 • **ValuePtr* がバイナリー・バッファーを指すポインターの場合、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *BufferLength* に入れます。それによって *BufferLength* は負の値になります。 • **ValuePtr* が文字ストリングまたはバイナリー・ストリング以外の値を指すポインターの場合、 *BufferLength* の値は SQL_IS_POINTER でなければなりません。 • **ValuePtr* に固定長のデータ・タイプが入っている場合、 *BufferLength* は SQL_IS_INTEGER か SQL_IS_UINTEGER (どちらか適切な方) になります。 • *ValuePtr* に入れられる戻り値が Unicode ストリングの場合、 *BufferLength* 引数は偶数でなければなりません。

表 99. SQLGetStmtAttr 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT *	*StringLengthPtr*	出力	**ValuePtr* に戻すために使用できる総バイト数 (NULL 終止符文字を除く) を戻すバッファーを指すポインター。このポインターが NULL ポインターの場合、長さは戻されません。属性値が文字ストリングで、戻りに使用できるバイト数が *BufferLength* 以上の場合、**ValuePtr* のデータは *BufferLength* から NULL 終止符文字の長さを減算した長さに切り捨てられ、CLI によってヌル終了になります。

使用法

SQLGetStmtAttr() を呼び出すと **ValuePtr* に、 *Attribute* に指定されているステートメント属性の値が戻されます。この値は、32 ビット値または NULL 文字で終了する文字ストリングのどちらかです。この値が NULL 文字で終了するストリングの場合、アプリケーションは *BufferLength* 引数にそのストリングの最大長を指定し、CLI は **StringLengthPtr* バッファーにそのストリングの長さを戻します。この値が 32 ビット値である場合、 *BufferLength* 引数と *StringLengthPtr* 引数は使用されません。

次の引数属性は読み取り専用なので、 SQLGetStmtAttr() で取り出しできますが、SQLSetStmtAttr() で設定はできません。設定および検索できるすべてのステートメント属性のリストの詳細は、ステートメント属性リストを参照してください。

- SQL_ATTR_IMP_PARAM_DESC
- SQL_ATTR_IMP_ROW_DESC
- SQL_ATTR_ROW_NUMBER

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 100. SQLGetStmtAttr SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	**ValuePtr* に戻されるデータは、 *BufferLength* から NULL 終止符文字の長さを引いた長さに切り捨てられます。 **StringLengthPtr* には、切り捨て前のストリング値の長さが戻されます。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
24000	カーソル状態が無効です。	引数 *Attribute* が SQL_ATTR_ROW_NUMBER で、カーソルがオープンされていなかったか、カーソルが結果セットの開始点の前または終了点の後ろに配置されていました。

表 100. SQLGetStmtAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	*StatementHandle* で非同期実行関数が呼び出されましたが、この関数が呼び出された時点でまだ実行中でした。 *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *BufferLength* に指定された値は、0 より小さい値でした。
HY092	オプション・タイプが範囲外です。	引数 *Attribute* に指定された値が、このバージョンの CLI では無効なものでした。
HY109	カーソルの位置が無効です。	*Attribute* 引数は SQL_ATTR_ROW_NUMBER でしたが、行は削除済みであったか、または取り出せませんでした。
HYC00	ドライバーが使用できません。	引数 *Attribute* で指定された値はこのバージョンの CLI の有効な CLI 属性でしたが、データ・ソースでサポートされていませんでした。

制限

なし。

例

```
/* get the handle for the implicitly allocated descriptor */
rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_IMP_ROW_DESC,
                    &hIRD,
                    SQL_IS_INTEGER,
                    &indicator);
```

SQLGetStmtOption 関数 (CLI) - ステートメント・オプションの現行設定値を戻す

ODBC 3.0 では SQLGetStmtOption() は使用すべきでない関数なので、代わりに SQLGetStmtAttr() を使用します。

このバージョンの CLI でも引き続き SQLGetStmtOption() をサポートしていますが、最新の標準に準拠するように、SQLGetStmtAttr() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

```
SQLGetStmtOption(hstmt, SQL_ATTR_CURSOR_HOLD, pvCursorHold);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLGetStmtAttr(hstmt, SQL_ATTR_CURSOR_HOLD, pvCursorHold,
               SQL_IS_INTEGER, NULL);
```

SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し

現行トランザクション中にサーバーから戻された (フェッチまたは直前の SQLGetSubString() 呼び出しによって戻された) ラージ・オブジェクト・ロケーターによって参照される、ラージ・オブジェクト値の一部を取り出します。

仕様:

- **CLI** 2.1

構文

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

関数引数

表 101. SQLGetSubString 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドルこれは、既に割り振り済みではあるが、現在は準備済み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・ハンドルです。
SQLSMALLINT	*LocatorCType*	入力	C タイプのソース LOB ロケーター。これは、次のいずれかです。 • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	入力	*Locator* は、ソース LOB ロケーター値に設定しなければなりません。
SQLUINTEGER	*FromPosition*	入力	BLOB と CLOB の場合、これは関数で戻される最初のバイトの位置です。 DBCLOB の場合、これは最初の文字です。最初のバイトまたは文字には、番号 1 が付けられます。

表 101. SQLGetSubString 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLUINTEGER	*ForLength*	入力	これは、関数から戻される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、これは文字単位の長さです。 *FromPosition* がソース・ストリングの長さより小さい値で、 *FromPosition* + *ForLength* - 1 がソース・ストリングの終わりを超えている場合、その結果は必要な文字数で右側を埋め込まれます (BLOB の場合は X'00'、 CLOB の場合は 1 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)。
SQLSMALLINT	*TargetCType*	入力	*DataPtr* の C データ・タイプ。ターゲットは常に、次のような LOB ロケーター C バッファー・タイプでなければなりません。 • SQL_C_CLOB_LOCATOR • SQL_C_BLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR または、次のような C ストリング・タイプでなければなりません。 • SQL_C_CHAR • SQL_C_WCHAR • SQL_C_BINARY • SQL_C_DBCHAR
SQLPOINTER	*DataPtr*	出力	取り出される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を指すポインター。
SQLINTEGER	*BufferLength*	入力	*DataPtr* で指示されたバッファーの最大サイズ (バイト単位)。
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であり、ロケーター値ではない場合に、 *DataPtr* に戻される情報の長さ (バイト数)[a]。 ポインターを null (NULL) に設定すると、何も返されません。
SQLINTEGER *	*IndicatorValue*	出力	常にゼロに設定されます。

注:

a これは、DBCLOB データの場合であってもバイト単位です。

使用法

SQLGetSubString() は、 LOB ロケーターで表されるストリングの部分を取得するときに使用します。ターゲットに関する選択項目には、次の 2 つがあります。

- ターゲットを、適切な C ストリング変数とすることができます。
- サーバーに新しい LOB 値を作成し、その値の LOB ロケーターをクライアント上のターゲット・アプリケーション変数に割り当てることができます。

SQLGetSubString() を SQLGetData の代わりに使用し、 LOB データを分割して取得することができます。この場合、まず列を LOB ロケーターにバインドし、次にそのロケーターを使用して LOB を全体でまたは分割してフェッチします。

Locator 引数には、任意の有効な LOB ロケーターを入れられます。ただし、そのロケーターは、FREE LOCATOR ステートメントを用いて明示的に解放されたり、またはロケーターの作成時にトランザクションが終了したために暗黙的に解放されたりしていないものとします。

このステートメント・ハンドルは、準備済みステートメントまたはカタログ関数呼び出しに関連付けられていてはなりません。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 102. SQLGetSubString SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	戻りデータが *BufferLength* より長くなっています。戻りに使用できるデータの実際の長さは、*StringLength* に保管されます。
07006	無効な変換です。	*TargetCType* に指定された値は、SQL_C_CHAR、 SQL_WCHAR、SQL_C_BINARY、 SQL_C_DBCHAR、または LOB ロケーターではありませんでした。 *TargetCType* に指定された値は、ソースにとって不適切です (例えば、BLOB 列に SQL_C_DBCHAR など)。
22011	サブストリング・エラーが起きました。	*FromPosition* が、ソース・ストリングの長さより大きくなっています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY003	プログラム・タイプが範囲外です。	*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	引数の値が無効です。	*FromPosition* または *ForLength* に指定した値は、正の整数ではありませんでした。

表 102. SQLGetSubString SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	指定した *StatementHandle* は、*割り振られて* いません。 実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	*BufferLength* の値は、0 より小さい値でした。
HYC00	ドライバーが使用できません。	アプリケーションは現在、ラージ・オブジェクトをサポートしないデータ・ソースに接続しています。
0F001	現在ロケーターは割り当てられていません。	*Locator* に指定した値は現在、LOB ロケーターではありません。

制限

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場合は、この関数は使用できません。関数タイプを SQL_API_SQLGETSUBSTRING に設定して SQLGetFunctions() を呼び出し、 *fExists* 出力引数を調べて、現行の接続でその関数がサポートされているかどうかを判別してください。

IDS データ・サーバーの場合、SQLGetSubstring() の *FromPosition* に、ゼロまたは負の値は指定できません。これは、現在の制限事項です。

IDS データ・サーバーにアクセスする場合に、 SQLGetSubString() 関数に SQL_C_CLOB_LOCATOR または SQL_C_BLOB_LOCATOR の TargetCType 引数値を指定して呼び出すと、「無効な変換」のエラーが返されます。このような変換は、サポートされていません。

例

```
/* read the piece of CLOB data in buffer */
cliRC = SQLGetSubString(hstmtLocUse,
                        SQL_C_CLOB_LOCATOR,
                        clobLoc,
                        clobPiecePos,
                        clobLen - clobPiecePos,
                        SQL_C_CHAR,
                        buffer,
                        clobLen - clobPiecePos + 1,
                        &clobPieceLen,
                        &ind);
```

SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得

CLI に関連した DBMS でサポートされるデータ・タイプに関する情報を戻します。

情報は、SQL 結果セット内に戻されます。照会を処理するときに使用される関数と同じ関数を使用して、列を受け取ることができます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

構文

```
SQLRETURN   SQLGetTypeInfo (
               SQLHSTMT         StatementHandle,  /* hstmt */
               SQLSMALLINT      DataType);        /* fSqlType */
```

関数引数

表 103. SQLGetTypeInfo 引数

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

表 103. SQLGetTypeInfo 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*DataType*	入力	照会される SQL データ・タイプ。サポートされるタイプは次のとおりです。 • SQL_ALL_TYPES • SQL_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_CHAR • SQL_CLOB • SQL_DATE [1] • SQL_TYPE_DATE • SQL_DBCLOB • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TIME [1] • SQL_TIMESTAMP [1] • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_TINYINT • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_XML SQL_ALL_TYPES の指定があると、サポートされるすべてのデータ・タイプに関する情報が、TYPE_NAME ごとに昇順で戻されます。サポートされないすべてのデータ・タイプは、結果セットには入れられません。

注:

1. これらの SQL データ・タイプは、 ODBC 2.0 との互換性がサポートされています。

使用法

SQLGetTypeInfo() は結果セットを生成し、照会を実行する場合と同じ処理を行うので、カーソルの生成やトランザクションの開始も行います。このステートメント・ハンドルで別のステートメントを作成して実行するには、カーソルをクローズする必要があります。

SQLGetTypeInfo() が無効な *DataType* を使用して呼び出されると、空の結果セットが戻されます。

LONGDATACOMPAT キーワードまたは SQL_ATTR_LONGDATA_COMPAT 接続属性のどちらかを設定すると、 *DATA_TYPE* 引数には SQL_BLOB、SQL_CLOB および SQL_DBCLOB の代わりに、 SQL_LONGVARBINARY と SQL_LONGVARCHAR および SQL_LONGVARGRAPHIC が戻されます。

この関数で生成される結果セットの列について、次のセクションで説明します。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。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データ・タイプです。ロケーター・データ・タイプなどの非持続性データ・タイプは、戻される結果セットには含まれません。ユーザー定義データ・タイプも戻されません。

SQLGetTypeInfo で戻される列

列 1 TYPE_NAME (VARCHAR(128) NOT NULL データ・タイプ)
データ・ソース依存のデータ・タイプ名。例えば "CHAR()"、"LONG VARBINARY" など。アプリケーションは、CREATE TABLE および ALTER TABLE ステートメント内でこの名前を使用しなければなりません。

列 2 DATA_TYPE (SMALLINT NOT NULL データ・タイプ)
SQL データ・タイプ定義値 (例えば、SQL_VARCHAR、 SQL_BLOB、SQL_DATE、SQL_INTEGER)。

列 3 COLUMN_SIZE (INTEGER データ・タイプ)
データ・タイプが文字またはバイナリー・ストリングの場合、この列にはバイト単位の最大長が入れられます。 GRAPHIC (DBCS) ストリングの場合、これはその列の 2 バイト文字の数になります (この動作は CLI/ODBC 構成キーワード Graphic によって変更可能)。データ・タイプが XML の場合、ゼロが戻されます。

日付、時刻、タイム・スタンプ・データ・タイプの場合、これは文字に変換されたときに値を表示するために必要となる文字数の合計です。

数値データ・タイプの場合は、これは桁数の合計 (精度) です。

列 4 LITERAL_PREFIX (VARCHAR(128) データ・タイプ)
DB2 がこのデータ・タイプのリテラルの接頭部として認識する文字。リテラル接頭部が適用外である場合には、データ・タイプのこの列は NULL です。

列 5 LITERAL_SUFFIX (VARCHAR(128) データ・タイプ)
DB2 がこのデータ・タイプのリテラルの接尾部として認識する文字。リテラル接頭部が適用外である場合には、データ・タイプのこの列は NULL です。

列 6 CREATE_PARAMS (VARCHAR(128) データ・タイプ)
この列のテキストには、 TYPE_NAME 列の名前を SQL のデータ・タイプとして使用したときにアプリケーションが括弧内に指定できる各パラメーターに対応付けて、キーワードをコンマで区切って列挙したリストが入ります。このリスト内のキーワードは、LENGTH、PRECISION、SCALE のいずれかにできます。これらは、SQL 構文で使用する際に必要とされる順序に従って配列されています。

データ・タイプ定義のためのパラメーター (例えば、INTEGER) がない場合は、 NULL 標識が戻されます。

注: CREATE_PARAMS の目的は、アプリケーションが *DDL ビルダー* のインターフェースをカスタマイズできるようにすることです。アプリケーションは、これを使用してできるのは、データ・タイプを定義したり、編集制御のラベルを付けるのに使用できるテキストをローカライズするために必要な引数の個数を判別することだけであることを予期する必要があります。

列 7 NULLABLE (SMALLINT NOT NULL データ・タイプ)
データ・タイプが NULL 値を受け入れるかどうかを指示します。
- NULL 値を使用できない場合は SQL_NO_NULLS に設定します。
- NULL 値を使用できる場合は SQL_NULLABLE に設定します。
- NULL 値を使用できるかどうか分からない場合は SQL_NULLABLE_UNKNOWN に設定します。

列 8 CASE_SENSITIVE (SMALLINT NOT NULL データ・タイプ)
照合と比較のときに文字データ・タイプで大文字小文字を区別するかどうかを指示します。有効な値は SQL_TRUE と SQL_FALSE です。

列 9 SEARCHABLE (SMALLINT NOT NULL データ・タイプ)
WHERE 節でのデータ・タイプの使用方法を示します。有効値は次のとおりです。
- SQL_UNSEARCHABLE : データ・タイプが WHERE 節で使用できない場合。
- SQL_LIKE_ONLY : WHERE 節で、データ・タイプが LIKE 述部でのみ使用できる場合。
- SQL_ALL_EXCEPT_LIKE : WHERE 節で、データ・タイプが LIKE を除くすべての比較演算子で使用できる場合。
- SQL_SEARCHABLE : WHERE 節で、データ・タイプがどの比較演算子ででも使用できる場合。

列 10 UNSIGNED_ATTRIBUTE (SMALLINT データ・タイプ)
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。この属性をデータ・タイプに適用できないと、NULL 標識が戻されます。

列 11 FIXED_PREC_SCALE (SMALLINT NOT NULL データ・タイプ)
データ・タイプが厳密な数であり、かつ常に同じ精度とスケールである場合には、値 SQL_TRUE が入ります。そうでない場合には、SQL_FALSE が入ります。

列 12 AUTO_INCREMENT (SMALLINT データ・タイプ)
行が挿入されたときに、このデータ・タイプの列が自動的にユニークな値に設定される場合には SQL_TRUE が入ります。そうでない場合には SQL_FALSE が入ります。

列 13 LOCAL_TYPE_NAME (VARCHAR(128) データ・タイプ)
この列には、データ・タイプの正規名とは異なる、データ・タイプのローカライズされた (ネイティブ言語の) 名前が入ります。ローカライズされた名前がない場合は、この列は NULL になります。

この列は表示専用です。ストリングの文字セットはロケールに依存しており、通常はデータベースのデフォルト文字セットです。

列 14 MINIMUM_SCALE (INTEGER データ・タイプ)
SQL データ・タイプの最小スケール。データ・タイプが固定スケールの場合、MINIMUM_SCALE 列と MAXIMUM_SCALE 列には同じ値が入れられます。スケールが適用できないと、NULL が戻されます。

列 15 MAXIMUM_SCALE (INTEGER データ・タイプ)
SQL データ・タイプの最大スケール。スケールが適用できないと、NULL が戻されます。最大スケールが DBMS 内で個別に定義されておらず、その代わりに列の最大長と同じものとして定義されている場合、この列には COLUMN_SIZE 列と同じ値が入れられます。

列 16 SQL_DATA_TYPE (SMALLINT NOT NULL データ・タイプ)
SQL_DESC_TYPE 記述子のフィールドに表示される SQL データ・タイプの値。この列は、 DATA_TYPE 列と同じです (CLI がサポートしていないインターバル・データ・タイプと日時データ・タイプは除く)。

列 17 SQL_DATETIME_SUB (SMALLINT データ・タイプ)
このフィールドは、常に NULL です (CLI はインターバル・データ・タイプと日時データ・タイプをサポートしていません)。

列 18 NUM_PREC_RADIX (INTEGER データ・タイプ)
データ・タイプが近似値タイプである場合、この列には値 2 が含まれており、これはビット数を COLUMN_SIZE で指定することを示しています。厳密な数値タイプである場合、この列には値 10 が含まれており、これは小数桁数を COLUMN_SIZE で指定することを示しています。その他の場合、この列は NULL になります。

列 19 INTERVAL_PRECISION (SMALLINT データ・タイプ)
このフィールドは、常に NULL です (CLI はインターバル・データ・タイプをサポートしていません)。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 104. SQLGetTypeInfo SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。 *StatementHandle* がクローズされていませんでした。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY004	SQL データ・タイプが範囲外です。	無効な *DataType* が指定されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

例

```
/* get data type information */
cliRC = SQLGetTypeInfo(hstmt, SQL_ALL_TYPES);
```

SQLMoreResults 関数 (CLI) - さらに結果セットがあるかどうかの判別

照会のパラメーター値の配列入力、結果セットを戻すストアード・プロシージャー、またはバッチ SQL に関連付けられているステートメント・ハンドルで入手できる情報がさらにあるかどうかを判別します。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

構文

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

関数引数

表 105. SQLMoreResults 引数

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

使用法

この関数は、以下の実行時に複数の結果セットを順次に戻すために使用されます。

- SQL_ATTR_PARAMSET_SIZE ステートメント属性および SQLBindParameter() で指定された入力パラメーター値の配列が指定されているパラメーター化照会、または
- SQL 照会を伴ったストアード・プロシージャー。ストアード・プロシージャーが実行を終えてもまだ結果セットにアクセスできるよう、この照会のカーソルはオープンされたままになっています。このシナリオの場合、ストアード・プロシージャーは通常、複数の結果セットを戻そうと試みます。
- またはバッチ SQL。単一の SQLExecute() または SQLExecDirect() の処理で、複数の SQL ステートメントがまとめて実行される場合。

最初の結果セットの処理が完了した後、アプリケーションは SQLMoreResults() を呼び出して、別の結果セットが利用できるかどうかを判別します。現在の結果セットがまだ取り出されていない行であれば、SQLMoreResults() はカーソルをクローズしてそれらを廃棄し、別の結果セットが利用できるなら、SQL_SUCCESS を戻します。

すべての結果セットが処理されると、SQLMoreResults() は SQL_NO_DATA_FOUND を戻します。

複数の結果セットを同時に操作できるようにする予定のアプリケーションは、CLI 関数 SQLNextResult() を呼び出せば、別のステートメント・ハンドルに結果セットを移動することができます。 SQLNextResult() はバッチ・ステートメントをサポートしません。

バッチ SQL の使用時には SQLExecute() または SQLExecDirect() は、そのバッチ内の最初の SQL ステートメントだけを実行します。その後で SQLMoreResults() を呼び出して次の SQL ステートメントを実行した場合、その次のステートメントの実行が正常に完了すると SQL_SUCCESS が戻されます。実行するステートメントがもうなくなった場合、SQL_NO_DATA_FOUND が戻されます。バッチ SQL ステートメントが UPDATE、INSERT、または DELETE ステートメントである場合、SQLRowCount() を呼び出せば、影響を受ける行の数を確かめることができます。

SQLCloseCursor() が呼び出された場合や、SQL_CLOSE オプションを指定して SQLFreeStmt() が呼び出された場合、あるいは *HandleType* を SQL_HANDLE_STMT に設定して SQLFreeHandle() が呼び出された場合、このステートメント・ハンドル上のすべてのペンディング結果セットは廃棄されます。

戻りコード

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

診断

表 106. SQLMoreResults SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

また、 SQLMoreResults() は SQLExecute() に関連した SQLSTATE を戻すことができます。

例

```
cliRC = SQLMoreResults(hstmt);
```

SQLNativeSql 関数 (CLI) - ネイティブ SQL テキストの取得

CLI がベンダー・エスケープ節を解釈する方法を表示します。

アプリケーションから渡された元の SQL ストリングにベンダー・エスケープ節シーケンス列が入っていた場合、 CLI はデータ・ソースによって参照される変換後の SQL ストリングを戻します。 (ベンダー・エスケープ節は適宜変換されるかまたは破棄されるかのどちらかです。)

仕様:

- **CLI** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLNativeSqlW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLNativeSql     (
               SQLHDBC           ConnectionHandle,  /* hdbc */
               SQLCHAR           *InStatementText,  /* szSqlStrIn */
               SQLINTEGER        TextLength1,       /* cbSqlStrIn */
               SQLCHAR           *OutStatementText, /* szSqlStr */
               SQLINTEGER        BufferLength,      /* cbSqlStrMax */
               SQLINTEGER        *TextLength2Ptr);  /* pcbSqlStr */
```

関数引数

表 107. SQLNativeSql 引数

データ・タイプ	引数	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* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLCHAR *	*OutStatementText*	出力	変換済み出力ストリングのバッファーを指すポインター。
SQLINTEGER	*BufferLength*	入力	*OutStatementText* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。
SQLINTEGER *	*TextLength2Ptr*	出力	*OutStatementText* に戻すために使用できる SQLCHAR エレメント (この関数の Unicode 版の場合は SQLWCHAR エレメント) の合計数 (NULL 終止符文字を除く)。戻り値に使用できる SQLCHAR エレメントの数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの数) が *BufferLength* 以上の場合、*OutStatementText* 内の出力 SQL ストリングは、 SQLCHAR または SQLWCHAR のエレメント数 *BufferLength* -1 個分に切り捨てられます。

使用法

CLI によってデータ・ソースに渡される変換済み SQL ストリングをアプリケーションで検査または表示する場合にこの関数を呼び出します。変換 (マッピング) は、入力 SQL ステートメント・ストリングにベンダー・エスケープ節シーケンス列が入っているときしか行われません。

SQLNativeSql() の呼び出し時には CLI は、ベンダーのエスケープ節の構文エラーしか検出できません。 CLI は、変換後の SQL ストリングをデータ・ソースに渡して準備できるようにすることはないので、構文エラーが DBMS によって検出されても、エラーはこの時点では生成されません。 (ステートメントが準備のためにデータ・ソースへ渡されないのは、その準備によりトランザクションが開始される可能性があるからです。)

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 108. SQLNativeSql SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	バッファー *OutStatementText* の容量が不足しており、 SQL ストリング全体を入れることができなかったため、ストリングを切り捨てました。引数 *TextLength2Ptr* に、切り捨てられていない SQL ストリングの全長が含まれています。 (関数は、SQL_SUCCESS_WITH_INFO で戻ります。)
08003	接続がクローズされています。	*ConnectionHandle* は、オープン・データベース接続を参照していません。
37000	SQL 構文が無効です。	*InStatementText* にある入力 SQL ストリングに構文エラーがありました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	引数 *InStatementText* は NULL ポインターです。 引数 *OutStatementText* は NULL ポインターです。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *TextLength1* は 0 より小さく、SQL_NTS と等しくありませんでした。 引数 *BufferLength* は、0 より小さい値でした。

制限

なし。

SQLNumParams 関数 (CLI) - SQL ステートメント内のパラメーター数の取得

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。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

構文

```
SQLRETURN   SQLNumParams (
               SQLHSTMT          StatementHandle,      /* hstmt */
               SQLSMALLINT       *ParameterCountPtr);  /* pcpar */
```

関数引数

表 109. SQLNumParams 引数

データ・タイプ	引数	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数。

使用法

Statement Handle に関連した準備済み SQL ステートメントがバッチ SQL (セミコロンで区切られた複数の SQL ステートメント) を備えている場合、ストリング全体のパラメーターがカウントされ、そのバッチを構成する個々のステートメント・ハンドル別にはカウントされません。

この関数は、 *StatementHandle* に関連したステートメントが準備された後でしか呼び出せません。ステートメントにパラメーター・マーカーが含まれていないと、*ParameterCountPtr* が 0 に設定されます。

アプリケーションは、この関数を呼び出して、このステートメント・ハンドルと関連した SQL ステートメント用に必要な `SQLBindParameter()` (または `SQLBindFileToParam()`) 呼び出しの数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

診断

表 110. SQLNumParams SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 `SQLCancel()` がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。

表 110. SQLNumParams SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	指定した *StatementHandle* で SQLPrepare() を呼び出す前に、この関数を呼び出しました。 実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

SQLNextResult 関数 (CLI) - 別のステートメント・ハンドルへの次の結果セットの関連付け

SQLNextResult() を使うと、ストアード・プロシージャーから戻された複数の結果セットに順不同でアクセスすることができます。

仕様:

- **CLI** 7.x

構文

```
SQLRETURN   SQLNextResult (SQLHSTMT StatementHandle1
                           SQLHSTMT StatementHandle2);
```

関数引数

表 111. SQLNextResult 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle1*	入力	ステートメント・ハンドル
SQLHSTMT	*StatementHandle2*	入力	ステートメント・ハンドル

使用法

ストアード・プロシージャーは、終了後も 1 つ以上のカーソルをオープンしたままにすることで、複数の結果セットを戻します。最初の結果セットにアクセスするには、そのストアード・プロシージャーを呼び出したステートメント・ハンドルを常に使います。複数の結果セットが戻された場合、SQLMoreResults() または SQLNextResult() を使って、結果セットの記述とフェッチを行うことができます。

SQLMoreResults() は、最初の結果セットのカーソルのクローズに使われるとともに、同じステートメント・ハンドルで次の結果セットを処理する手段にもなるのに対して、 SQLNextResult() は、*StatementHandle1* 上のカーソルをクローズしないまま、次の結果セットを *StatementHandle2* に移動します。フェッチする結果セットがない場合、どちらの関数も SQL_NO_DATA_FOUND を戻します。

SQLNextResult() を使用した場合、他のステートメント・ハンドルに転送された後の結果セットを任意の順序で処理することができます。 *StatementHandle1* 上にもうカーソル (オープンされた結果セット) がなくなるまで、 SQLMoreResults() と SQLNextResult() を混合して呼び出すことができます。

SQLNextResult() が SQL_SUCCESS を戻すと、次の結果セットはもう *StatementHandle1* には関連付けられていません。つまり、SQLExecDirect() が *StatementHandle2* に対する照会を正常に完了したばかりであるかのように、次の結果セットは *StatementHandle2* に関連付けられています。したがって、SQLNumResultCols()、SQLDescribeCol()、または SQLColAttribute() を使ってカーソルを記述できるということです。

SQLNextResult() の呼び出しが完了すると、それまで *StatementHandle2* に関連付けられていた結果セットは、残りの結果セットのチェーンから除去されるので、SQLNextResult() または SQLMoreResults() で再使用することはできません。すなわち、n 個の結果セットがある場合、最大 n-1 回だけ SQLNextResult() を正常に呼び出せるということです。

`SQLCloseCursor()` が呼び出された場合や、SQL_CLOSE オプションを指定して SQLFreeStmt() が呼び出された場合、あるいは *HandleType* を SQL_HANDLE_STMT に設定して SQLFreeHandle() が呼び出された場合、このステートメント・ハンドル上のすべてのペンディング結果セットは廃棄されます。

StatementHandle2 にオープン・カーソルがある場合や、 *StatementHandle1* と *StatementHandle2* が同じ接続上にない場合、SQLNextResult() は SQL_ERROR を戻します。エラーまたは警告が戻された場合は常に、*StatementHandle1* で `SQLGetDiagRec()` を呼び出さなければなりません。

注: SQLMoreResults() は、 SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性および SQLBindParameter() を使って指定された入力パラメーター値の配列が指定されているパラメーター化照会とも連動して稼働します。ただし SQLNextResult() はこれをサポートしていません。

戻りコード

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

診断

表 112. SQLNextResult SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	想定外のシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。
HY010	関数のシーケンス・エラー。	実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 *StatementHandle2* には、関連したオープン・カーソルがあります。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	想定外のメモリー処理エラーです。	DB2 CLI は、この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アクセスできませんでした。
HYT00	タイムアウトの有効期限切れです。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

パラメーター化照会およびバッチ SQL に使用できるのは、SQLMoreResults() だけです。

例

```
/* use SQLNextResult to push Result Set 2 onto the second statement handle */
cliRC = SQLNextResult(hstmt, hstmt2); /* open second cursor */
```

SQLNumResultCols 関数 (CLI) - 結果列の数の取得

入力ステートメント・ハンドルに関連した結果セット内の列数を戻します。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

この関数を呼び出す前に、 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、SQLColAttribute()、またはいずれかのバインド列関数を呼び出すことができます。

構文

```
SQLRETURN   SQLNumResultCols (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLSMALLINT       *ColumnCountPtr);  /* pccol */
```

関数引数

表 113. SQLNumResultCols 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLSMALLINT *	*ColumnCountPtr*	出力	結果セット内の列の数。

使用法

この関数は、入力ステートメント・ハンドルに関して実行された最後のステートメントまたは関数が結果セットを生成しなかった場合に、出力引数をゼロに設定します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 114. SQLNumResultCols SQLSTATEs

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	SQLPrepare() または SQLExecDirect() を *StatementHandle* 用に呼び出す前に、この関数が呼び出されました。 実行時データ (SQLParamData()、 SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。

表 114. SQLNumResultCols SQLSTATEs (続き)

SQLSTATE	説明	解説
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

許可

なし。

例

```
/* identify the number of output columns */
cliRC = SQLNumResultCols(hstmt, &nResultCols);
```

SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得

SQLPutData() とともに、いくつかの部分に分かれた長いデータを送信します。また、実行時に固定長データを送るときにも使用することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0
- **ISO CLI**

構文

```
SQLRETURN   SQLParamData (
               SQLHSTMT         StatementHandle, /* hstmt */
               SQLPOINTER       *ValuePtrPtr );  /* prgbValue */
```

関数引数

表 115. SQLParamData 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLPOINTER *	*ValuePtrPtr*	出力	SQLBindParameter() (パラメーター・データの場合) に指定されている *ParameterValuePtr* バッファーのアドレス、または SQLBindCol() (列データの場合) に指定されている *TargetValuePtr* バッファーのアドレスを戻すためのバッファーを指すポインター。これらのアドレスは、SQL_DESC_DATA_PTR 記述子レコード・フィールドにあります。
		入力	バージョン 9.7 フィックスパック 1 以降、SQL_ATTR_INTERLEAVED_PUTDATA が TRUE に設定されると、これは入力引数になります。アプリケーションは、後続の SQLPutData() 呼び出しでデータを PUT するための値を提供します。

使用法

SQLParamData() は、データがまだ割り当てられていない 1 つ以上の SQL_DATA_AT_EXEC パラメーターが存在する場合に SQL_NEED_DATA を戻します。この関数は、直前の SQLBindParameter() 呼び出し時にアプリケーションが提供した *ValuePtrPtr* の値を戻します。 SQLPutData() は、パラメーター・データを送信するため (長いデータの場合) 1 回以上にわたり呼び出されます。
SQLParamData() は、現行パラメーターに関するすべてのデータが送られたことを知らせるときと、次の SQL_DATA_AT_EXEC パラメーターに進むときに呼び出します。 SQL_SUCCESS は、すべてのパラメーターにデータ値が割り当てられ、関連したステートメントが正常に実行されたときに戻されます。実際のステートメント実行時か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 を戻すと、 SQLPutData() または SQLCancel() の呼び出しだけを行うことができます。このステートメント・ハンドルを使用する他の関数呼び出しはすべて失敗します。さらに、*StatementHandle* の親接続ハンドルを参照するすべての関数呼び出しに、属性や接続状態を変更する処理が関係している場合、これらの呼び出しは失敗します。つまり、親接続ハンドルに対する以下の関数も許可されません。

- SQLSetConnectAttr()
- SQLEndTran()

しかし、完了タイプとして SQL_ROLLBACK を指定して SQLEndTran() 関数への呼び出しを行うことは、SQL_ATTR_FORCE_ROLLBACK 接続属性が設定され、StreamPutData 構成キーワードが 1 に設定され、自動コミット・モードが有効になっている場合に許可されます。

以下の関数が SQL_NEED_DATA シーケンス時に呼び出されると、 SQLSTATE **HY**010 の SQL_ERROR が戻され、 SQL_DATA_AT_EXEC パラメーターの処理は影響を受けません。

戻りコード

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

診断

SQLParamData() は、SQLPrepare()、SQLExecDirect()、および SQLExecute() 関数から戻された SQLSTATE を戻すことができます。また、次の診断を生成することもできます。

表 116. SQLParamData SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	CLI とアプリケーション変数の間でデータを転送すると、非互換のデータ変換が行われます。

表 116. SQLParamData SQLSTATE (続き)

SQLSTATE	説明	解説
22026	ストリング・データ、長さの不一致	SQLGetInfo() の SQL_NEED_LONG_DATA_LEN 情報タイプは 'Y' でしたが、長いパラメーター (データ・タイプが SQL_LONGVARCHAR、SQL_LONGVARBINARY、その他の長いデータ・タイプ) 用に送られたデータは、 SQLBindParameter() で *StrLen_or_IndPtr* 引数を使用して指定された値よりも短いデータでした。 SQLGetInfo() の SQL_NEED_LONG_DATA_LEN 情報タイプは 'Y' でしたが、長い列 (データ・タイプが SQL_LONGVARCHAR、SQL_LONGVARBINARY、その他の長いデータ・タイプ) 用に送られたデータは、 SQLSetPos() を使用して更新されたデータの行の列に対応する長さバッファーに指定された値よりも短いデータでした。
40001	トランザクションのロールバック	この SQL ステートメントが属するトランザクションは、デッドロックまたはタイムアウトが原因でロールバックされました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY000	一般エラーです。	特定の SQLSTATE が用意されておらず、しかもインプリメンテーション独自の SQLSTATE も定義されていないエラーが発生しました。 SQLGetDiagRec() から引数 *MessageText* 内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	SQLParamData() を順不同で呼び出しました。この呼び出しは、SQLExecDirect() または SQLExecute() の後か、 SQLPutData() 呼び出しの後だけ有効です。 SQLExecDirect() または SQLExecute() 呼び出しの後にこの関数を呼び出しましたが、処理する SQL_DATA_AT_EXEC パラメーターがありません (残っていません) でした。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY092	オプション・タイプが範囲外です。	直前の SQLBindFileToParam() 操作の *FileOptions* 引数が無効でした。
HY506	ファイルのクローズ・エラーです。	一時ファイルをクローズしようとしているときにエラーが発生しました。
HY509	ファイルの削除エラーです。	一時ファイルを削除しようとしているときにエラーが発生しました。

表 116. SQLParamData SQLSTATE (続き)

SQLSTATE	説明	解説
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

例

```
/* get next parameter for which a data value is needed */
cliRC = SQLParamData(hstmt, (SQLPOINTER *)&valuePtr);
```

SQLParamOptions 関数 (CLI) - パラメーターの入力配列の指定

ODBC 3.0 では SQLParamOptions() は使用すべきでない関数なので、代わりに SQLSetStmtAttr() を使用します。

このバージョンの CLI でも引き続き SQLParamOptions() をサポートしていますが、最新の標準に準拠するように、SQLSetStmtAttr() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

```
SQLParamOptions(hstmt, crow, pirow);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLSetStmtAttr(hstmt, fOption, pvParam, fStrLen);
```

SQLPrepare 関数 (CLI) - ステートメントの準備

SQLPrepare() は、SQL ステートメントまたは XQuery 式を、 提供された入力ステートメント・ハンドルに関連付けます。

アプリケーションは、その SQL ステートメント内に 1 つ以上のパラメーター・マーカーを組み込むことができます。アプリケーションがパラメーター・マーカーを組み込むには、SQL ストリングの適切な位置に疑問符 (?)、またはコロンの後に名前が続く形 (*:name*) を組み込みます。アプリケーションは、ステートメント・ハンドルを他の関数に渡して、この準備済みステートメントを参照することができます。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

注: XQuery 式の場合、式そのものにパラメーター・マーカーを指定することはできません。しかし、XMLQUERY 関数を使用して、パラメーター・マーカーを XQuery 変数にバインドすることができます。次に、バインド済みパラメーター・マーカーの値は、実行のために、XMLQUERY で指定された XQuery 式に渡されます。

すでに照会ステートメント (または結果セットを戻す任意の関数) でステートメント・ハンドルを使用している場合、 SQLPrepare() を呼び出す前に、 SQL_CLOSE オプションを指定した SQLCloseCursor() または 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。

XQuery 式の前には "XQUERY" キーワードを付ける必要があります。このキーワードを含めなくても XQuery 式を準備して実行するには、SQLPrepare() または SQLExecDirect() を呼び出す前に、ステートメント属性 SQL_ATTR_XQUERY_STATEMENT を SQL_TRUE に設定してください。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLPrepareW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

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

関数引数

表 117. SQLPrepare 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル *StatementHandle* に関連したオープン・カーソルがあってはなりません。
SQLCHAR *	*StatementText*	入力	SQL ステートメント・ストリング。
SQLINTEGER	*TextLength*	入力	*StatementText* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *StatementText* がヌル終了ストリングの場合は SQL_NTS。

使用法

据え置き準備は、デフォルトでオンになっています。準備済みステートメントと同じステートメント・ハンドルを使って SQLDescribeParam()、 SQLExecute()、 SQLNumResultCols()、 SQLDescribeCol()、または SQLColAttribute() を呼び出さないかぎり、PREPARE 要求はサーバーに送られません。これによってネットワーク・フローが最小限になり、パフォーマンスが向上します。

SQL ステートメント・テキストにベンダー・エスケープ節シーケンス列が入っている場合、 CLI は、準備のために SQL ステートメント・テキストをデータベースにサブミットする前に、まず SQL ステートメント・テキストを適切な DB2 特定フォ

ーマットに修正します。アプリケーションがベンダー・エスケープ節シーケンスを備えた SQL を生成しない場合は、 SQL_ATTR_NOSCAN ステートメント属性を接続レベルで SQL_NOSCAN に設定することによって、 CLI がどのベンダー・エスケープ節に対してもスキャンを実行しないようにしなければなりません。

SQLPrepare() を使用してステートメントをいったん準備したら、アプリケーションは次のものを呼び出して、結果セット (照会ステートメントであった場合) のフォーマットに関する情報を要求することができます。

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttribute()

StatementText 内のパラメーター・マーカーに関する情報を要求するには、以下の関数を使用します。

- SQLDescribeParam()
- SQLNumParams()

注: SQLNumResultCols()、SQLDescribeCol()、SQLColAttribute()、または SQLDescribeParam() を初めて呼び出した場合は、据え置き準備が使用できるなら、PREPARE 要求がサーバーに強制的に送信されます。

SQL ステートメント・ストリングには、パラメーター・マーカーが含まれている場合があり、 SQLNumParams() を呼び出して、ステートメント内のパラメーター・マーカーの個数を判別することができます。パラメーター・マーカーは ? 文字、またはコロンの後に名前が続く形 (*:name*) で表されます。これは、SQLExecute() の呼び出し時にステートメント内のどの位置でアプリケーション提供の値を置き換えるかを示すために使用されます。バインド・パラメーター関数である SQLBindParameter()、SQLSetParam()、および SQLBindFileToParam() を使用するのは、アプリケーションの変数を各パラメーター・マーカーにバインドする (関連付ける) 場合と、データの転送時にデータの変換を実行する必要があるかどうかを指示する場合です。アプリケーションは SQLDescribeParam() を呼び出して、データベース・サーバーに送られる予定のパラメーター・マーカーのデータに関する情報を検索することができます。

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おく必要があります。

パラメーター・マーカーに関する規則の詳細は、PREPARE ステートメントの項を参照してください。

アプリケーションが SQLExecute() 呼び出しからの結果を処理した後で、新しい (または同じ) パラメーター値で再度ステートメントを実行することができます。

その SQL ステートメントとしては COMMIT または ROLLBACK が可能です。また、このステートメントのどちらかを実行すると、現在の接続ハンドルで SQLEndTran() を呼び出すのと同じ効果を生じます。

SQL ステートメントが定位置 DELETE または定位置 UPDATE の場合は、そのステートメントで参照されるカーソルを、同じ接続ハンドルおよび同じ分離レベルの個別のステートメント・ハンドルで定義する必要があります。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 118. SQLPrepare SQLSTATE

SQLSTATE	説明	解説
01504	UPDATE または DELETE ステートメントに、WHERE 節がありません。	*StatementText* の UPDATE ステートメントまたは DELETE ステートメントに、 WHERE 節が入っていませんでした。
01508	ブロッキングには不適格なステートメントです。	このステートメントは、ストレージ以外の理由でブロッキングできませんでした。
21S01	挿入値リストが列リストに一致していません。	*StatementText* に INSERT ステートメントがあり、挿入する値の数が派生表の数と一致していませんでした。
21S02	派生表の次数が列リストに一致していません。	*StatementText* に CREATE VIEW ステートメントがあり、指定された名前の数は、照会指定で定義されている派生表と同じ数になっていません。
22018	キャスト指定の文字値が無効です。	*StatementText* にリテラルまたはパラメーターを含む SQL ステートメントがあり、この値には関連した表の列のデータ・タイプとの互換がありませんでした。
22019	無効なエスケープ文字	引数 *StatementText* の WHERE 節に ESCAPE 付きの LIKE 述部があり、 ESCAPE の後に続くエスケープ文字の長さが 1 と等しくありませんでした。
22025	無効なエスケープ・シーケンス	引数 *StatementText* の WHERE 文節に「LIKE *パターン値* ESCAPE *エスケープ文字*」があり、パターン値のエスケープ文字の後の文字は「%」でも「_」でもありませんでした。
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
34000	カーソル名が無効です。	*StatementText* に、位置指定された DELETE または位置指定された UPDATE がありますが、実行中のステートメントで参照されているカーソルはオープンされていませんでした。
37xxx [a]	SQL 構文が無効です。	*StatementText* に、以下のうちの 1 つ以上の問題が含まれていました。 • 接続されたデータベース・サーバーが準備できない SQL ステートメント • 構文エラーのあるステートメント
40001	トランザクションのロールバック	この SQL ステートメントが属するトランザクションは、デッドロックまたはタイムアウトが原因でロールバックされました。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。

表 118. SQLPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
42xxx [a]	構文エラーまたはアクセス規則違反	**42**5xx は、この許可 ID が、*StatementText* に含まれている SQL ステートメントを実行する権限を持っていないことを示します。 他の **42**xxx SQLSTATE は、構文の相違またはステートメントとのアクセス問題があることを示しています。
58004	予期しないシステム障害です。	回復不能システム・エラー。
S0001	データベース・オブジェクトはすでにあります。	*StatementText* 内に、CREATE TABLE ステートメントまたは CREATE VIEW ステートメントがあり、指定されている表名またはビュー名はすでに存在しています。
S0002	データベース・オブジェクトはありません。	*StatementText* に、存在していない表名またはビュー名を参照する SQL ステートメントがあります。
S0011	索引はすでにあります。	*StatementText* に CREATE INDEX ステートメントがあり、指定された索引名はすでに存在していました。
S0012	索引は見つかりません。	*StatementText* に DROP INDEX ステートメントがあり、指定された索引名は存在していませんでした。
S0021	列はすでにあります。	*StatementText* 内には ALTER TABLE ステートメントがありますが、 ADD 節に指定されている列はユニークな列ではなかったか、または基本表の既存の列を識別していました。
S0022	列は見つかりません。	*StatementText* に、存在していない列名を参照する SQL ステートメントがあります。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	*StatementText* は、NULL ポインターでした。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *TextLength* は 1 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

表 118. SQLPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
注:		
a	xxx は、そのクラス・コードの任意の SQLSTATE を表します。例えば、 **37**xxx は **37** クラスの任意の SQLSTATE を表します。	

注: すべての DBMS が準備時に上記の診断メッセージをすべて報告するわけではありません。据え置き準備がデフォルトの振る舞い (SQL_ATTR_DEFERRED_PREPARE ステートメント属性で制御します) としてオンのままになっていると、 PREPARE がサーバーに送られたときにこのようなエラーが発生する可能性があります。アプリケーションは、このような流れを生じる関数を呼び出すときにこれらの条件を処理できなければなりません。この種の関数には、SQLExecute()、SQLDescribeParam()、SQLNumResultCols()、SQLDescribeCol()、および SQLColAttribute() などがあります。

許可

なし。

例

```
SQLCHAR *stmt = (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? ";

/* ... */

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

SQLPrimaryKeys 関数 (CLI) - 表の主キー列の取得

SQLPrimaryKeys() 関数は、表の主キーを構成する列名のリストを戻します。

この情報は SQL 結果セットにして戻されます。これは、照会により生成された結果セットの処理で使用するものと同じ関数を用いて取り出すことができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

SQLPrimaryKeys() は、表の主キーを構成する列名のリストを戻します。情報は SQL 結果セット内に戻されますが、照会により作成された結果セットを処理するのに使用される関数と同じ関数を使用して情報を取り出すことができます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLPrimaryKeysW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLPrimaryKeys   (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szCatalogName */
```

```
SQLSMALLINT       NameLength1,       /* cbCatalogName */
SQLCHAR           *SchemaName,       /* szSchemaName */
SQLSMALLINT       NameLength2,       /* cbSchemaName */
SQLCHAR           *TableName,        /* szTableName */
SQLSMALLINT       NameLength3);      /* cbTableName */
```

関数引数

表 119. SQLPrimaryKeys 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	表名のスキーマ修飾子。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名。
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。

使用法

SQLPrimaryKeys() 関数は、シングル表から主キー列を戻します。どの引数の指定にも、検索パターンは使用できません。

結果セットには、SQLPrimaryKeys で戻される列にリストされている列が、TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および ORDINAL_POSITION 列の順序で含まれています。

多くの場合に SQLPrimaryKeys() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

スキーマ名を指定しないと、現行接続で有効なものがデフォルトのスキーマ名になります。

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_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、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLPrimaryKeys で戻される列

列 1 TABLE_CAT (VARCHAR(128))
主キー表カタログ名。この表にカタログがない場合、この値は NULL になります。

列 2 TABLE_SCHEM (VARCHAR(128))
TABLE_NAME を含むスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
指定した表の名前。

列 4 COLUMN_NAME (VARCHAR(128) 非 NULL)
主キー列名。

列 5 KEY_SEQ (SMALLINT 非 NULL)
1 を最初の番号とする主キー内の列シーケンス番号。

列 6 PK_NAME (VARCHAR(128))
主キー ID。データ・ソースに適用できない場合は、NULL。

注: CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の SQLPrimaryKeys() 結果セットで定義されているものと同じです。

指定した表に主キーが含まれていない場合、空の結果セットが戻されます。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 120. SQLPrimaryKeys SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYC00	ドライバーが使用できません。	CLI は、表名の修飾子として *catalog* をサポートしません。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

なし。

例

```
/* get the primary key columns of a table */
cliRC = SQLPrimaryKeys(hstmt, NULL, 0, tbSchema, SQL_NTS, tbName, SQL_NTS);
```

SQLProcedureColumns 関数 (CLI) - プロシージャーの入出力パラメーター情報の取得

SQLProcedureColumns() 関数は、ストアード・プロシージャーに関連する入出力パラメーターのリストを戻します。

この情報は SQL 結果セットで返されます。この結果セットは、照会によって生成される結果セットを処理するために使用するのと同じ関数を使用して取得することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLProcedureColumnsW() です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN SQLProcedureColumns(
            SQLHSTMT         StatementHandle,   /* hstmt */
            SQLCHAR          *CatalogName,      /* szProcCatalog */
            SQLSMALLINT      NameLength1,       /* cbProcCatalog */
            SQLCHAR          *SchemaName,       /* szProcSchema */
            SQLSMALLINT      NameLength2,       /* cbProcSchema */
            SQLCHAR          *ProcName,         /* szProcName */
            SQLSMALLINT      NameLength3,       /* cbProcName */
            SQLCHAR          *ColumnName,       /* szColumnName */
            SQLSMALLINT      NameLength4);      /* cbColumnName */
```

関数引数

表 121. SQLProcedureColumns 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための *パターン値* が入れられるバッファー。 DB2 for z/OS の場合、ストアード・プロシージャーは 1 つのスキーマにあります。*SchemaName* 引数の唯一の受け入れ可能な値は NULL ポインターです。値を指定しても、空の結果セットと SQL_SUCCESS が戻されます。 DB2 Database for Linux, UNIX, and Windows 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。有効な検索パターンの詳細は、カタログ関数の入力引数の項を参照してください。

表 121. SQLProcedureColumns 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ProcName*	入力	プロシージャー名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength3*	入力	*ProcName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ProcName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ColumnName*	入力	パラメーター名で結果セットを修飾するための *パターン値* が入れられるバッファー。この引数を使用するのは、 ProcName、SchemaName、またはその両方に空でない値を指定することによって既に限定されている結果セットをさらに修飾する場合です。
SQLSMALLINT	*NameLength4*	入力	*ColumnName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ColumnName* がヌル終了ストリングの場合は SQL_NTS。

使用法

SQLProcedureColumns() 関数は、
PROCEDURE_CAT、PROCEDURE_SCHEM、PROCEDURE_NAME、および COLUMN_TYPE の順序で結果セット内の情報を戻します。SQLProcedureColumns で戻される列は、結果セット内の列をリストしています。アプリケーションは、将来のリリースで最終列以降の列が定義される可能性があることを考慮しておく必要があります。

多くの場合に SQLProcedureColumns() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、および COLUMN_NAME 列の実際の長さを判別することができます。

SQL_ATTR_LONGDATA_COMPAT 接続属性が設定されている場合、LOB 列タイプは LONG VARCHAR、LONG VARBINARY、または LONG VARGRAPHIC タイプとして報告されます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

ストアード・プロシージャーが DB2 for z/OS サーバーにある場合、そのストアード・プロシージャーの名前をサーバーの SYSIBM.SYSPROCEDURES カタログ表に登録しなければなりません。V8 およびそれ以降のサーバーの場合、そのサーバーの SYSIBM.SYSROUTINES および SYSIBM.SYSPARAMS カタログ表にストアード・プロシージャーを登録しなければなりません。

ストアード・プロシージャー・カタログの機能を提供しない他の DB2 サーバーのバージョンでは、空の結果セットが戻されます。

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、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に関する記述子情報を戻すことはできません。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

SQLProcedureColumns で戻される列

列 1 PROCEDURE_CAT (VARCHAR(128))
プロシージャー・カタログ名。このプロシージャーにカタログがない場合、この値は NULL になります。

列 2 PROCEDURE_SCHEM (VARCHAR(128))
PROCEDURE_NAME を含むスキーマの名前。DB2 for z/OS の SQLProcedureColumns() 結果セットの場合は NULL です。

列 3 PROCEDURE_NAME (VARCHAR(128))
プロシージャーの名前。

列 4 COLUMN_NAME (VARCHAR(128))
パラメーターの名前。

列 5 COLUMN_TYPE (SMALLINT 非 NULL)
この行に関連した情報のタイプを識別します。戻される値は以下のとおりです。

- SQL_PARAM_INPUT は入力パラメーターです。
- SQL_PARAM_INPUT_OUTPUT は入出力パラメーターです。
- SQL_PARAM_OUTPUT は出力パラメーターです。

ODBC 仕様で定義されていても戻されない値は以下のとおりです。

- SQL_PARAM_TYPE_UNKNOWN : パラメーター・タイプが不明です。
- SQL_RETURN_VALUE はプロシージャー列内のプロシージャーの戻り値です。
- SQL_RESULT_COL は、結果セット内の列です。

列 6 DATA_TYPE (SMALLINT 非 NULL)
SQL データ・タイプ。

列 7 TYPE_NAME (VARCHAR(128) 非 NULL)
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。

列 8 COLUMN_SIZE (INTEGER)
SQL ルーチン内の XML 引数の場合、(XML 引数には長さがないため) ゼロが戻されます。しかし、カタログ式外部ルーチンの場合、XML パラメーターは XML AS CLOB(n) として宣言されます。この場合、COLUMN_SIZE はカタログされた長さ、n です。

DATA_TYPE 列の値が文字ストリングまたはバイナリー・ストリングを示す場合、この列には SQLCHAR または SQLWCHAR エレメント数で表記した最大長が入れられます。DATA_TYPE 列の値が GRAPHIC (DBCS) ストリングの場合、COLUMN_SIZE はパラメーターの 2 バイトの SQLCHAR エレメントまたは SQLWCHAR エレメントの数になります。

日付、時刻、およびタイム・スタンプのデータ・タイプの場合は、文字データ・タイプに変換された場合に値を表示するために必要な SQLCHAR エレメントまたは SQLWCHAR エレメントの数の合計です。

数値データ・タイプの場合、COLUMN_SIZE 値は結果セット内の NUM_PREC_RADIX 列の値に基づいて、列に許可されている総桁数または合計ビット数のいずれかです。

データ・タイプ精度の表を参照してください。

列 9 BUFFER_LENGTH (INTEGER)
SQL_C_DEFAULT が `SQLBindCol()`、`SQLGetData()` および `SQLBindParameter()` 呼び出しで指定された場合に、このパラメーターからのデータを保管するための関連する C バッファーの最大バイト。この長さには、NULL 終止符は含まれません。厳密な数データ・タイプを出すには、長さとして小数部や符号も考慮されます。

SQL ルーチン内の XML 引数の場合、(XML 引数には長さがないため) ゼロが戻されます。しかし、カタログ式外部ルーチンの場合、XML パラメーターは XML AS CLOB(n) として宣言されます。この場合、BUFFER_LENGTH はカタログされた長さ、n です。

データ・タイプ長の表を参照してください。

列 10 DECIMAL_DIGITS (SMALLINT)
パラメーターのスケール。スケールが適用できないデータ・タイプの場合は NULL が戻されます。

データ・タイプ・スケールの表を参照してください。

列 11 NUM_PREC_RADIX (SMALLINT)
10、2、または NULL のいずれか。DATA_TYPE が近似値データ・タイプである場合、この列には値 2 が含まれており、COLUMN_SIZE 列にはそのパラメーターに入れられるビット数が含まれています。

DATA_TYPE が高精度数値データ・タイプの場合、この列には値 10 が含まれており、COLUMN_SIZE 列と DECIMAL_DIGITS 列にはそのパラメーターに入れられる小数桁数が含まれています。

数値データ・タイプの場合、DBMS は 10 または 2 の NUM_PREC_RADIX を戻すことができます。

基数が適用できないデータ・タイプの場合は NULL が戻されます。

列 12 NULLABLE (SMALLINT 非 NULL)
パラメーターが NULL を受け入れない場合は SQL_NO_NULLS。

パラメーターが NULL 値を受け入れる場合は SQL_NULLABLE。

列 13 REMARKS (VARCHAR(254))
パラメーターに関する記述情報を入れられます。

列 14 COLUMN_DEF (VARCHAR)
列のデフォルト値。

NULL をデフォルト値として指定した場合、この列は引用符で囲まれていない語 NULL です。切り捨てを行わないとデフォルト値を表すことができない場合、この列には単一引用符で囲まれていない TRUNCATED が入ります。デフォルト値を指定しなかった場合、この列は NULL です。

COLUMN_DEF の値は、TRUNCATED 以外の値であれば新しい列定義を生成するために使用できます。

列 15 SQL_DATA_TYPE (SMALLINT 非 NULL)
SQL_DESC_TYPE 記述子のフィールドに表示される SQL データ・タイプの値。日時データ・タイプは除いて、この列は DATA_TYPE 列と同じです (CLI はインターバル・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 (SQL_CODE_DATE、SQL_CODE_TIME、または SQL_CODE_TIMESTAMP) のサブコードを戻します。

列 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 値を戻します (CLI がサポートしていないインターバル・データ・タイプを含む)。

列 17 CHAR_OCTET_LENGTH (INTEGER)
文字データ・タイプ列のバイト単位の最大長。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。

列 18 ORDINAL_POSITION (INTEGER NOT NULL)
この結果セットの COLUMN_NAME で指定されているパラメーターの序数部が入れられます。これは、CALL ステートメント上で提供される引数の元の位置です。左端の引数の元の位置は、1 です。

列 19 IS_NULLABLE (Varchar)
- 列に NULL を含めることができない場合は、「NO」。
- 列に NULL が含まれる場合は、「YES」。
- NULL 可能かどうか不明の場合は、ゼロ長ストリング。

NULL 可能かどうかを判別する際には、ISO 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。)

注:

- CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の SQLProcedureColumns() 結果セットで定義されているものと同じです。
- 2 つのモジュールに同じ名前のプロシージャーが組み込まれている場合、SQLProcedureColumns() は両方のプロシージャーに関する詳細を戻します。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 122. SQLProcedureColumns SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
42601	PARMLIST 構文エラーです。	ストアード・プロシージャー・カタログ表の PARMLIST 値に、構文エラーがあります。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。

表 122. SQLProcedureColumns SQLSTATE (続き)

SQLSTATE	説明	解説
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

SQLProcedureColumns() 関数は、ストアード・プロシージャーから戻された可能性のある結果セットの属性についての情報を戻しません。

ストアード・プロシージャー・カタログのサポートや、ストアード・プロシージャーのサポートを提供していない DB2 にアプリケーションを接続すると、SQLProcedureColumns() 関数は空の結果セットを戻します。

例

```
/* get input/output parameter information for a procedure */
sqlrc = SQLProcedureColumns(hstmt,
                            NULL,
                            0, /* catalog name not used */
                            (unsigned char *)colSchemaNamePattern,
                            SQL_NTS, /* schema name not currently used */
                            (unsigned char *)procname,
                            SQL_NTS,
                            colNamePattern,
                            SQL_NTS); /* all columns */
```

SQLProcedures 関数 (CLI) - プロシージャー名のリストの取得

SQLProcedures() 関数は、サーバーに登録されていてしかも指定した検索パターンと一致するストアード・プロシージャー名のリストを戻します。

この情報は SQL 結果セットで返されます。この結果セットは、照会によって生成される結果セットを処理するため使用するのと同じ関数を使用して取得することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLProceduresW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLProcedures    (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szProcCatalog */
               SQLSMALLINT       NameLength1,       /* cbProcCatalog */
               SQLCHAR           *SchemaName,       /* szProcSchema */
```

```
SQLSMALLINT       NameLength2,       /* cbProcSchema */
SQLCHAR           *ProcName,         /* szProcName */
SQLSMALLINT       NameLength3);      /* cbProcName */
```

関数引数

表 123. SQLProcedures の引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための *パターン値* が入れられるバッファー。 DB2 for z/OS の場合、ストアード・プロシージャーは 1 つのスキーマにあります。*SchemaName* 引数の唯一の受け入れ可能な値は NULL ポインターです。値を指定しても、空の結果セットと SQL_SUCCESS が戻されます。 DB2 Database for Linux, UNIX, and Windows 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。有効な検索パターンの詳細は、カタログ関数の入力引数の項を参照してください。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*ProcName*	入力	表名で結果セットを修飾するための *パターン値* を入れられるバッファー。
SQLSMALLINT	*NameLength3*	入力	*ProcName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *ProcName* がヌル終了ストリングの場合は SQL_NTS。

使用法

SQLProcedures() 関数から戻された結果セットには、所定の順序で SQLProcedures で戻される列にリストされている列が含まれています。これらの行は、PROCEDURE_CAT、PROCEDURE_SCHEMA、および PROCEDURE_NAME の順序になります。

多くの場合に SQLProcedures() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_SCHEMA_NAME_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、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SQL_ATTR_LONGDATA_COMPAT 接続属性が設定されている場合、LOB 列タイプは LONG VARCHAR、LONG VARBINARY、または LONG VARGRAPHIC タイプとして報告されます。

ストアード・プロシージャーが DB2 for z/OS サーバーにある場合、そのストアード・プロシージャーの名前をサーバーの SYSIBM.SYSPROCEDURES カタログ表に登録しなければなりません。V8 およびそれ以降のサーバーの場合、そのサーバーの SYSIBM.SYSROUTINES および SYSIBM.SYSPARAMS カタログ表にストアード・プロシージャーを登録しなければなりません。

ストアード・プロシージャー・カタログの機能を提供しない DB2 サーバーの他のバージョンでは、空の結果セットが戻されます。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLProcedures で戻される列

列 1 PROCEDURE_CAT (VARCHAR(128))
プロシージャー・カタログ名。このプロシージャーにカタログがない場合、この値は NULL になります。

列 2 PROCEDURE_SCHEM (VARCHAR(128))
PROCEDURE_NAME を含むスキーマの名前。

列 3 PROCEDURE_NAME (VARCHAR(128) NOT NULL)
プロシージャーの名前。

列 4 NUM_INPUT_PARAMS (INTEGER 非 NULL)
入力パラメーター数。INOUT パラメーターは、この数の中にカウントされません。

INOUT パラメーターの詳細を確かめるには、SQLProcedureColumns() から戻される COLUMN_TYPE 列を調べてください。

列 5 NUM_OUTPUT_PARAMS (INTEGER 非 NULL)
出力パラメーター数。INOUT パラメーターは、この数の中にカウントされません。

INOUT パラメーターの詳細を確かめるには、SQLProcedureColumns() から戻される COLUMN_TYPE 列を調べてください。

列 6 NUM_RESULT_SETS (INTEGER 非 NULL)
プロシージャーから戻される結果セット数。

この列は、今後 ODBC で使用するために予約済みになっているので、使用しないでください。

列 7 REMARKS (VARCHAR(254))
プロシージャーに関する記述情報が含まれています。

列 8 PROCEDURE_TYPE (SMALLINT)
プロシージャー・タイプを次のように定義します。
- SQL_PT_UNKNOWN: プロシージャーが値を戻すかどうかを判別することはできません。
- SQL_PT_PROCEDURE: 戻されるオブジェクトはプロシージャーで、戻り値がありません。
- SQL_PT_FUNCTION: 戻されるオブジェクトは関数で、戻り値があります。

CLI は常に SQL_PT_PROCEDURE を戻します。

注:

- CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列の名前、内容、および順序は、ODBC の SQLProcedures() 結果セットで定義されているものと同じです。
- 2 つのモジュールに同じ名前のプロシージャーが組み込まれている場合、SQLProcedures() 関数は両方のプロシージャーに関する詳細を戻します。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 124. SQLProcedures SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。

表 124. SQLProcedures SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。この関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

ストアード・プロシージャー・カタログのサポートや、ストアード・プロシージャーのサポートを提供していない DB2 にアプリケーションを接続すると、SQLProcedureColumns() は空の結果セットを戻します。

SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し

大きなパラメーター値を分割して送ります。 SQLPutData() を呼び出した後、SQLParamData() が呼び出され、 SQL_NEED_DATA を戻してパラメーター・データ値を渡します。

仕様:

- **CLI** 2.1
- **ODBC** 1.0
- **ISO CLI**

構文

```
SQLRETURN   SQLPutData      (
              SQLHSTMT          StatementHandle,   /* hstmt */
              SQLPOINTER        DataPtr,           /* rgbValue */
              SQLLEN            StrLen_or_Ind);    /* cbValue */
```

関数引数

表 125. SQLPutData 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLPOINTER	*DataPtr*	入力	パラメーターに関する実際のデータまたはデータの一部を指すポインター。データは、パラメーターを指定するときにアプリケーションが使用した、SQLBindParameter() 呼び出しで指定されている書式でなければなりません。
SQLLEN	*StrLen_or_Ind*	入力	*DataPtr* の長さ。呼び出しで送られるデータの量を SQLPutData() に指定します。 データ量は、指定パラメーターでの呼び出しごとに違う場合があります。アプリケーションは、*StrLen_or_Ind* に SQL_NTS または SQL_NULL_DATA を指定することもできます。 *StrLen_or_Ind* は、データ、時間、タイム・スタンプのようなすべての固定長 C バッファー・タイプ、およびすべての数値 C バッファー・タイプに対して無視されます。 C バッファー・タイプが SQL_C_CHAR または SQL_C_BINARY である場合、または SQL_C_DEFAULT を C バッファー・タイプとして指定し、 C バッファー・タイプのデフォルト値が SQL_C_CHAR または SQL_C_BINARY である場合、これは *DataPtr* バッファー内のデータのバイト数です。

使用法

アプリケーションは、SQL_NEED_DATA 状態のときに SQLParamData() を呼び出した後でステートメント上の SQLPutData() を呼び出して、 SQL_DATA_AT_EXEC パラメーターのデータ値を渡します。 SQLPutData() 呼び出しを繰り返して、長いデータを分割して送ることができます。 CLI は各 SQL_DATA_AT_EXEC パラメーターごとに一時ファイルを生成し、 SQLPutData() の呼び出し時にデータが一部ずつそこに付加されます。 CLI が一時ファイルを作成する場所であるパスを設定するには、 db2cli.ini ファイル内の TEMPDIR キーワードを使用します。このキーワードが設定されないと CLI は、環境変数 TEMP または TMP で指定されたパスへの書き込みを試みます。パラメーター・データのすべての部分を送った後、アプリケーションは SQLParamData() を再度呼び出して次の SQL_DATA_AT_EXEC に進むか、または、すべてのパラメーターにデータ値がある場合はステートメントを実行します。

SQLPutData() を SQL_C_LONG のような固定長 C バッファー・タイプに対して 2 回以上呼び出すことはできません。

入力データが文字データまたはバイナリー・データの場合、 SQLPutData() 呼び出しの後で有効な関数呼び出しは SQLParamData()、SQLCancel()、または別の SQLPutData() だけです。 SQLParamData() の場合と同様に、このステートメント・ハンドルを使用して他の関数呼び出しを行うと、すべて失敗します。さらに、*StatementHandle* の親接続ハンドルを参照する関数呼び出しの場合はすべて、属性や接続状態を変更する処理が関係していると、この呼び出しは失敗します。つまり、親接続ハンドルに対する以下の関数も許可されません。

- SQLSetConnectAttr()
- SQLEndTran()

しかし、完了タイプとして SQL_ROLLBACK を指定して SQLEndTran() 関数への呼び出しを行うことは、SQL_ATTR_FORCE_ROLLBACK 接続属性が設定され、StreamPutData 構成キーワードが 1 に設定され、自動コミット・モードが有効になっている場合に許可されます。

以下の関数が SQL_NEED_DATA シーケンス時に呼び出されると、 SQLSTATE **HY**010 の SQL_ERROR が戻され、 SQL_DATA_AT_EXEC パラメーターの処理は影響を受けません。

シングル・パラメーターについて 1 回以上 SQLPutData() 呼び出しを行って SQL_SUCCESS になった場合、同じパラメーターについて *StrLen_or_Ind* を SQL_NULL_DATA に設定して SQLPutData() を呼び出そうとすると、 SQLSTATE **22**005 のエラーが発生します。このエラーでは、状態は変化しません。つまり、ステートメント・ハンドルはまだ *Need Data* 状態で、アプリケーションはパラメーター・データの送信を続けます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

次の診断条件の中には、SQLPutData() を呼び出す時点ではなく最後の SQLParamData() 呼び出しの時点で報告されるものがあります。

表 126. SQLPutData SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	数値パラメーターに送られるデータは、有効数字が失われないようにして切り捨てられます。 送信された日付と時刻の列に関するタイム・スタンプ・データが切り捨てられました。 関数は、 SQL_SUCCESS_WITH_INFO で戻ります。

表 126. SQLPutData SQLSTATE (続き)

SQLSTATE	説明	解説
22001	ストリング・データの右側が切り捨てられました。	バイナリー・データまたは文字データに、その列でデータ・ソースがサポートできるサイズより大きなデータが送られました。
22003	数値が範囲外です。	送られた数値パラメーターのデータが原因で、関連する列への割り当て時に数値の整数部分が切り捨てられました。 固定長パラメーターについて SQLPutData() を複数回呼び出しました。
22005	割り当てにエラーがありました。	パラメーターに送られたデータには、関連した表の列のデータ・タイプとの互換がありませんでした。
22007	無効な日付時刻形式です。	日付、時刻、またはタイム・スタンプのパラメーターに送られたデータ値は、無効でした。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	引数 *DataPtr* が NULL ポインターで、引数 *StrLen_or_Ind* は 0 でも SQL_NULL_DATA でもありませんでした。
HY010	関数のシーケンス・エラーです。	ステートメント・ハンドル *StatementHandle* は、データを必要としている状態でなければならず、直前の SQLParamData() 呼び出しによって SQL_DATA_AT_EXEC パラメーターに入れておく必要があります。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *DataPtr* は NULL ポインターではなく、引数 *StrLen_or_Ind* は、0 未満でしたが、 SQL_NTS または SQL_NULL_DATA と等しくありませんでした。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

StrLen_or_Ind 用のさらに別の値 SQL_DEFAULT_PARAM が ODBC 2.0 に導入されたので、アプリケーションから送信された値ではなく、パラメーターのデフォルト値をプロシージャーで使用することを指定できるようになりました。 DB2 ストアード・プロシージャー引数ではデフォルト値はサポートされないので、StrLen_or_Ind 引数にこの値を指定すると、 SQL_DEFAULT_PARAM 値は無効な長さとみなされ、 CALL ステートメントを実行したときにエラーになります。

ODBC 2.0 には、*StrLen_or_Ind 引数* を指定して使用する SQL_LEN_DATA_AT_EXEC(*length*) マクロも導入されました。このマクロは、後続の SQLPutData() 呼び出しを経由して文字またはバイナリー C データ用に送信されるデータ全体の長さの合計を指定するために使用されます。 DB2 ODBC ドライバーではこの情報の必要がないため、このマクロは必要ありません。 ODBC アプリケーションは、 SQL_NEED_LONG_DATA_LEN オプションを指定した SQLGetInfo() を呼び出して、ドライバーがこの情報を必要とするかどうかを調べます。 DB2 ODBC ドライバーは、 SQLPutData() にはこの情報は必要ないことを示す 'N' を戻します。

例

```
SQLCHAR buffer[BUFSIZ];
size_t n = BUFSIZ;

/* ... */

/* passing data value for a parameter */
cliRC = SQLPutData(hstmt, buffer, n);
```

SQLReloadConfig 関数 (CLI) - クライアント構成ファイルから構成プロパティーを再ロードする

SQLReloadConfig() 関数は、db2dsdriver.cfg クライアント構成ファイルから構成プロパティーを再ロードします。

仕様:

- **CLI** 9.7

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLReloadConfigW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLReloadConfig(SQLINTEGER       ConfigProperty,
                            SQLCHAR          *DiagInfoString,
                            SQLSMALLINT      BufferLength,
                            SQLSMALLINT      *StringLengthPtr);
```

関数引数

表 127. SQLReloadConfig 関数引数

データ・タイプ	引数	使用法	説明
SQLINTEGER	*ConfigProperty*	入力	再ロードする db2dsdriver.cfg ファイル・セクションの事前定義済みグループ。サポートされる値は **DSD_ACR_AFFINITY** のみです。 **DSD_ACR_AFFINITY** 値は、使用法のセクションで定義されています。
SQLCHAR *	*DiagInfoString*	出力	値が SQL_ERROR の場合、CLI は **DiagInfoString** 出力に詳細なエラーの説明または情報を返します。値が SQL_SUCCESS または SQL_SUCCESS_WITH_INFO の場合、CLI は情報を返しません。 DB2 バージョン 9.7 フィックスパック 5 以降のフィックスパックでは、警告メッセージに、データベース名、サーバー名、およびポート番号から構成される診断ストリングの接頭部が付きます (*database*:*hostname*:*port*)。複数の警告は、改行文字で区切られます。

表 127. SQLReloadConfig 関数引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER	*BufferLength*	入力	**DiagInfoString* 引数の長さ。
SQLINTEGER *	*StringLengthPtr*	出力	**DiagInfoString* 引数に戻すために使用できる SQLCHAR エレメントの合計数 (この関数の Unicode 版の場合は SQLWCHAR エレメントの合計数) を戻すバッファーを指すポインター (NULL 終止符文字分のバイト数を除く)。戻されるバイト数が *BufferLength* 引数の値より大きい場合、 **DiagInfoString* 引数内のテキストは *BufferLength* 引数の値から NULL 終止符文字の長さを減算した長さに切り捨てられます。その後 CLI はヌル文字を使用して **DiagInfoString* 引数内のテキストを終了します。

使用法

db2dsdriver.cfg ファイルの変更後に、SQLReloadConfig() 関数を発行して、*ConfigProperty* 引数内で指定した db2dsdriver.cfg ファイルのセクションの項目を再ロードできます。

現在再ロードできるのは **DSD_ACR_AFFINITY** 構成プロパティーのみです。**DSD_ACR_AFFINITY** 構成プロパティーは以下のパラメーターで構成されます。これらのパラメーターは、db2dsdriver.cfg ファイル内の自動クライアント・リルート (<acr>) セクションで定義します。

- <alternateserverlist>
- <affinitylist>
- <clientaffinitydefined>
- <clientaffinityroundrobin>

<acr> セクション内で他のパラメーターに変更を加えても無視されます。

アプリケーションで SQLReloadConfig() 関数が発行されて再ロードが行われると、*次のトランザクション間隔* ですべての接続に関するアフィニティー・メンバーが再評価されます。次のトランザクション間隔とは、次のトランザクション境界のことを指し、コミットまたはロールバックの際に定義されます。アフィニティー・メンバーの再評価には、<acr> セクション内のすべての代替サーバーに関する項目の妥当性検査が含まれます。各サーバーに対して、指定されたホスト名とポート番号を使用して、ソケットのオープンが試行されます。アクティブ・データベース接続の代替サーバーのリスト内のサーバーがすべて到達不能な場合、SQLReloadConfig() 関数の **DiagInfoString** 引数に次のエラー・メッセージが返されます。

```
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - None of the servers,
   specified under <alternateserverlist> section, are reachable.
```

アイドル接続に関するアフィニティーは、その接続がアクティブになった時点で初めて評価されます。接続がアクティブになると、アフィニティー・メンバーに移動されます。

db2dsdriver.cfg ファイルの <acr> セクション以外のセクションを変更すると、SQLReloadConfig() 関数は *DiagInfoString* 引数内に SQL_ERROR の値を戻します。また、db2dsdriver.cfg ファイル内の既存のデータベース項目に関する <acr> セクションを削除したり追加したりすると、SQLReloadConfig() 関数はエラーを戻します。SQLReloadConfig() 関数がエラーを戻した場合、アプリケーションは古い db2dsdriver.cfg ファイル内容へのアクセスを続行します。

SQLReloadConfig 関数 (CLI) - クライアント構成ファイルから構成プロパティーを再ロードする

SQLReloadConfig() 関数は、メモリー内バージョンの自動クライアント・リルート (ACR) アフィニティーを、現行の db2dsdriver.cfg ファイル内でリストされているすべてのデータベースで更新します。いずれかのデータベース項目の <acr> セクションが無効であることがこの関数により検出された場合、そのデータベースへの接続が次に試みられたときに、エラーが発生します。無効な <acr> セクションが検出されると、次のトランザクション間隔で、アクティブ・データベースに対する接続も終了されます。<acr> セクションが無効になる理由として、以下の原因が考えられます。

- <database> セクションが欠落している (SQL_ERROR が戻されます)。
- <acr> セクションが欠落している (SQL_ERROR が戻されます)。
- 新しい代替サーバー・リストが空である。
- CLI が、新しいサーバー・リストを保持するための内部バッファー境界の制限を検出した。

SQLReloadConfig() 関数には、代替サーバー・ポートに対してソケットを開こうとする試行に関するデフォルトのタイムアウト値はありません。db2dsdriver.cfg ファイルのグローバル・セクション内で **SocketTimeOut** パラメーターを使用して、特定のタイムアウト値を設定できます。

戻りコード

- SQL_ERROR
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

以下の表に、SQLReloadConfig() 関数の呼び出し時に発生することがあるエラーと警告を示します。

表 128. SQLReloadConfig() 関数のエラー・メッセージ

エラー・メッセージ	解説
IBM DB2 [CLI Driver] 別の SQLReloadConfig() が既に実行中です。(IBM DB2 [CLI Driver] Another SQLReloadConfig() execution is already under progress.)	特定のプロセスに関する SQLReloadConfig() 関数が既に呼び出されており、実行中です。
IBM DB2 [CLI Driver] SQLReloadConfig() の ConfigProperty 引数で、認識されていない値が見つかりました。(IBM DB2 [CLI Driver] Unrecognized value found in ConfigProperty argument of SQLReloadConfig().)	**DSD_ACR_AFFINITY** 構成プロパティー以外の値が指定されました。
IBM DB2 [CLI Driver] 再ロードする db2dsdriver.cfg ファイルが、予期された場所で見つかりません。(IBM DB2 [CLI Driver] db2dsdriver.cfg file to be reloaded cannot be located in the expected location.)	db2dsdriver.cfg ファイルにアクセスできません。db2dsdriver.cfg ファイルが存在し、グローバルな読み取り権限があることを確認してください。
IBM DB2 [CLI ドライバー] (IBM DB2 [CLI Driver]) ConfigProperty 引数に指定されているセクション以外に、変更されているセクションが見つかりました。他のセクションの変更はサポートされていません (Sections other than one specified in ConfigProperty argument is found modified, which is not supported.)	<acr> セクション以外のセクションが変更されました。

表 128. SQLReloadConfig() 関数のエラー・メッセージ (続き)

エラー・メッセージ	解説
IBM DB2 [CLI Driver] CLI サブシステムが初期化されていません。(IBM DB2 [CLI Driver] CLI subsystem is not initialized.) SQLAllocHandle() を使用して環境ハンドルを割り振ってください。(Use SQLAllocHandle() to allocate environment handle.)	SQLReloadConfig() 関数を呼び出す前に環境ハンドルを割り振らなければなりません。
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - db2dsdriver.cfg 内に <client>、<affinitylist>、<alternateserverlist> セクションのすべてまたはいずれかがありません。(IBM DB2 [CLI Driver] <database>:<hostname>:<port> - Either all or one of <client>, <affinitylist>, <alternateserverlist> sections are missing in db2dsdriver.cfg.)	db2dsdriver.cfg ファイルで、 <client>、<affinitylist>、<alternateserverlist> セクションのいずれかまたはすべてに関する項目が欠落しています。
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - <alternateserverlist> セクションで指定したサーバーが到達不能です。(IBM DB2 [CLI Driver] <database>:<hostname>:<port> - None of the servers, specified under <alternateserverlist> section, are reachable.)	IBM ドライバーは、db2dsdriver.cfg ファイルの <alternateserverlist> セクション内で指定されているすべてのサーバーへの接続を確立できません。
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - サービス名 <srvcname> に該当するポート番号が見つかりません。(IBM DB2 [CLI Driver] <database>:<hostname>:<port> - cannot find appropriate port number for service name <srvcname>.)	IBM ドライバーは、エラー・メッセージで指定されているサービス名に関するポート番号を割り振ることができません。

表 129. SQLReloadConfig() 関数の警告メッセージ

エラー・メッセージ	解説
IBM DB2 [CLI Driver] <database>:<hostname>:<port> - <alternateserverlist> 内のサーバー「<hostname>:<port>、<hostname>:port、...」が到達不能でした。(IBM DB2 [CLI Driver] <database>:<hostname>:<port> - The following server(s) from <alternateserverlist> were unreachable"<hostname>:<port>,<hostname>:port,...".)	IBM ドライバーは、db2dsdriver.cfg ファイルの <alternateserverlist> セクション内で指定された一部のサーバーへの接続を確立できません。

制限

SQLReloadConfig() 関数を使用して再ロードできるプロパティーは、db2dsdriver.cfg ファイルの **DSD_ACR_AFFINITY** 構成プロパティーのみです。

SQLRowCount 関数 (CLI) - 行カウントの取得

表または表に基づいたビューに対して発行された UPDATE、INSERT、DELETE、または MERGE ステートメントの影響を受けた表中の行数を戻します。

仕様:

- **CLI** 1.1
- **ODBC** 1.0
- **ISO CLI**

SQLRowCount() は、表または表に基づいたビューに対して発行された UPDATE、INSERT、DELETE、または MERGE ステートメントの影響を受けた表中の行数を戻します。

SQLRowCount() を呼び出す前に、 SQLExecute() または SQLExecDirect() を呼び出す必要があります。

構文

```
SQLRETURN   SQLRowCount (
               SQLHSTMT     StatementHandle,   /* hstmt */
               SQLLEN       *RowCountPtr);     /* pcrow */
```

関数引数

表 130. SQLRowCount 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLLEN *	*RowCountPtr*	出力	影響を受けた行数が保管されるロケーションを指すポインター。 SQL_ATTR_PARC_BATCH 接続属性が SQL_PARC_BATCH_ENABLE に設定されると、ロケーション・サイズは配列のサイズにする必要があります。詳しくは、「使用法」セクションを参照してください。

使用法

入力ステートメント・ハンドルで参照されるステートメントのうち最後に発行されたものが UPDATE、INSERT、DELETE、または MERGE ステートメントでなかった場合、あるいは、そのステートメントを正常に実行できなかった場合、関数は *StatementHandle* の内容を -1 に設定します。

SELECT 専用順方向カーソルに SQLRowCount() 関数を使用した場合、この関数は *RowCountPtr* の内容を -1 に設定します。すべてのデータがフェッチされるまで行数は利用できません。CLI ステートメント属性 SQL_ATTR_ROWCOUNT_PREFETCH を使用して、データのフェッチ前にクライアントが完全な行カウントを要求できるようにすることが可能です。

制約事項: SQL_ATTR_ROWCOUNT_PREFETCH 属性は、カーソルに LOB または XML が含まれている場合はサポートされません。

SQL_ATTR_PARC_BATCH 接続属性が SQL_PARC_BATCH_ENABLE に設定されている場合、SQL_ATTR_PARAMOPT_ATOMIC 属性を SQL_ATOMIC_NO に設定し、*RowCountPtr* 引数が型 SQLLEN の配列を指すようにする必要があります。この配列の長さは、SQL_ATTR_PARAMSET_SIZE に等しくする必要があります。非アトミックの更新、削除、または挿入操作を正常に実行したときに、各パラメーター・セットによって影響を受けた表の行数は、この配列に格納されます。

そのステートメントによって影響を受けた (例えば、カスケード削除による) 可能性がある他表の行はすべて、このカウントから除外されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 131. SQLRowCount SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	*StatementHandle* で SQLExecute() または SQLExecDirect() を呼び出す前に、この関数を呼び出しました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。

許可

なし。

SQLSetColAttributes 関数 (CLI) - 列属性の設定

ODBC 3.0 では SQLSetColAttributes() は使用すべきでない関数なので、 CLI はこの関数をサポートしていません。

現在では、CLI はデフォルトで据え置き準備を使用するため、SQLSetColAttributes() の機能を必要としません。

SQLSetConnectAttr 関数 (CLI) - 接続属性の設定

接続の各局面を管理する属性を設定します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLSetConnectAttrW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN   SQLSetConnectAttr (
               SQLHDBC          ConnectionHandle,  /* hdbc */
               SQLINTEGER       Attribute,         /* fOption */
               SQLPOINTER       ValuePtr,          /* pvParam */
               SQLINTEGER       StringLength);     /* fStrLen */
```

関数引数

表 132. SQLSetConnectAttr 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLINTEGER	*Attribute*	入力	接続属性リストに載っている設定対象の属性。
SQLPOINTER	*ValuePtr*	入力	*Attribute* と関連付けられる値を指すポインター。 *ValuePtr* は、 *Attribute* の値に応じて 32 ビットの符号なし整数値またはヌル終了文字ストリングを指すポインターのどちらかになります。 *Attribute* 引数がドライバー固有の値である場合、 **ValuePtr* の値は符号付き整数でも構わないことに注意してください。詳細は、接続属性リストを参照してください。
SQLINTEGER	*StringLength*	入力	*Attribute* が ODBC 定義の属性で、 *ValuePtr* が文字ストリングかバイナリー・バッファーを指している場合、この引数の長さは **ValuePtr* の長さにする必要があります。文字ストリング・データの場合、 *StringLength* の内容は、そのストリングのバイト数でなければなりません。 *Attribute* が ODBC 定義の属性で、 *ValuePtr* が整数の場合、*StringLength* は無視されます。 *Attribute* が CLI 属性の場合、アプリケーションは *StringLength* 引数を設定して、その属性の性質を示します。 *StringLength* には以下の値が入ります。 • *ValuePtr* が文字ストリングを指すポインターの場合、*StringLength* は、そのストリングを格納するのに必要なバイト数、または SQL_NTS です。 • *ValuePtr* がバイナリー・バッファーを指すポインターの場合、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *StringLength* に入れます。それによって *StringLength* は負の値になります。 • *ValuePtr* が文字ストリングまたはバイナリー・ストリング以外の値を指すポインターの場合、 *StringLength* の値は SQL_IS_POINTER でなければなりません。 • *ValuePtr* に固定長の値が入っている場合、 *StringLength* は SQL_IS_INTEGER か SQL_IS_UINTEGER (どちらか適切な方) になります。

使用法

SQLSetConnectAttr() を使用したステートメント属性の設定に対するサポートの廃止

SQLSetConnectAttr() を使用してステートメント属性を設定する機能はサポートされなくなりました。バージョン 5 以前に作成されたアプリケーションをサポートするには、このリリースの CLI の SQLSetConnectAttr() を使用して、いくつかのステー

トメント属性を設定できます。しかし、この動作に依存するすべてのアプリケーションは、代わりに SQLSetStmtAttr() を使用して更新する必要があります。

SQLSetConnectAttr() を呼び出して記述子のヘッダー・フィールドを設定するステートメント属性を設定すると、その接続上のすべてのステートメントと現在関連しているアプリケーション記述子に合わせて記述子フィールドが設定されます。しかし、この属性設定は、今後この接続にあるステートメントと関連する可能性のある記述子には影響を与えません。

接続属性

アプリケーションは、接続が割り当てられてから解放されるまでの間に、いつでも SQLSetConnectAttr() を呼び出すことができます。アプリケーションが接続に正常に設定したすべての接続属性とステートメント属性は、この接続上で SQLFreeHandle() が呼び出されるまで有効です。

接続が設定される前しか設定できない接続属性もあります。接続が設定された後しか設定できない接続属性もあり、ステートメントが割り当てられてしまうと設定できないものもあります。各属性をいつ設定できるかの詳細は、接続属性リストを参照してください。

データ・ソースが *ValuePtr* に指定された値をサポートしない場合に、類似した値の置換をサポートする接続属性もあります。このような場合、CLI は SQL_SUCCESS_WITH_INFO と SQLSTATE 01S02 (オプション値が変更されました。) を戻します。アプリケーションは SQLGetConnectAttr() を呼び出して、代用された値を判別します。

ValuePtr を使って設定する情報のフォーマットは、指定する *Attribute* によって異なります。 SQLSetConnectAttr() は、2 つの異なるフォーマット、つまり、ヌル終了文字ストリングか、32 ビット整数値のどちらかのフォーマットで、属性情報を受け入れます。それぞれのフォーマットの注記は、その属性の説明にあります。SQLSetConnectAttr() の *ValuePtr* 引数が指す文字ストリングの長さは、 *StringLength* バイトになります。この長さが属性で定義されている場合、*StringLength* 引数は無視されます。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

CLI は、SQL_SUCCESS_WITH_INFO を戻して、オプション設定の結果に関する情報を提供することができます。

Attribute がステートメント属性の場合、SQLSetConnectAttr() は SQLSetStmtAttr() によって戻される任意の SQLSTATE を戻すことができます。

表 133. SQLSetConnectAttr SQLSTATE

SQLSTATE	説明	解説
01000	一般エラーです。	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S02	オプション値が変更されました。	CLI は、**ValuePtr* で指定した値をサポートしておらず、類似した値を代用しました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08002	接続が使用中です。	引数 *Attribute* は SQL_ATTR_ODBC_CURSORS で、 CLI はすでにデータ・ソースに接続されていました。
08003	接続がクローズされています。	オープン接続が必要な *Attribute* 値が指定されましたが、*ConnectionHandle* は接続状態になっていませんでした。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
24000	カーソル状態が無効です。	引数 *Attribute* は SQL_ATTR_CURRENT_QUALIFIER で、結果セットはペンディングになっていました。
HY000	一般エラーです。	特定の SQLSTATE が用意されておらず、しかもインプリメンテーション独自の SQLSTATE も定義されていないエラーが発生しました。 SQLGetDiagRec() から *MessageText バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	*ValuePtr* に NULL ポインターが渡され、**ValuePtr* の値はストリング値でした。
HY010	関数のシーケンス・エラーです。	*ConnectionHandle* と関連する *StatementHandle* で非同期関数が呼び出され、 SQLSetConnectAttr() が呼び出された時点でまだ実行中でした。 *ConnectionHandle* と関連する *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。 *ConnectionHandle* で SQLBrowseConnect() が呼び出され、SQL_NEED_DATA が戻されました。 SQLBrowseConnect() が SQL_SUCCESS_WITH_INFO または SQL_SUCCESS を戻す前にこの関数が呼び出されました。
HY011	この時点で無効な操作です。	引数 *Attribute* は SQL_ATTR_TXN_ISOLATION で、トランザクションはオープンされていました。

表 133. SQLSetConnectAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY024	属性の値が無効です。	指定されている *Attribute* 値に対して、**ValuePtr* に無効値を指定しました。 (CLI がこの SQLSTATE を戻すのは、SQL_ATTR_ACCESS_MODE などの離散的な値セットを受け入れる接続およびステートメント属性に対してのみです。その他すべての接続およびステートメント属性の場合、CLI は *ValuePtr* に指定されている値を検証する必要があります。) *Attribute* 引数は SQL_ATTR_TRACEFILE または SQL_ATTR_TRANSLATE_LIB で、 **ValuePtr* は空ストリングでした。
HY090	ストリングまたはバッファーの長さが無効です。	*StringLength* 引数は 0 より小さい値でしたが、SQL_NTS ではありませんでした。
HY092	オプション・タイプが範囲外です。	引数 *Attribute* に指定された値が、このバージョンの CLI では無効なものでした。
HYC00	ドライバーが使用できません。	引数 *Attribute* に指定された値は、このバージョンの CLI ドライバーには有効な接続またはステートメント属性でしたが、データ・ソースではサポートされていませんでした。

制限

なし。

例

```
/* set AUTOCOMMIT on */
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_AUTOCOMMIT,
                          (SQLPOINTER)SQL_AUTOCOMMIT_ON,
                          SQL_NTS);
/* ... */

/* set AUTOCOMMIT OFF */
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_AUTOCOMMIT,
                          (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
                          SQL_NTS);
```

SQLSetConnection 関数 (CLI) - 接続ハンドルの設定

この関数は、アプリケーションが実行を続ける前に特定の接続に決定的に切り替える必要がある場合に必要です。この関数を使用してもよいのは、アプリケーションが CLI 関数呼び出しと組み込み SQL 関数呼び出しを混合している場合に、複数の接続が関与しているときだけです。

仕様:

- **CLI** 2.1

構文

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

関数引数

表 134. SQLSetConnection 引数

データ・タイプ	引数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	アプリケーションが切り替えようとしている先の接続に関連した接続ハンドル。

使用法

CLI バージョン 1 では、CLI 接続関数を使って接続要求を出した場合にかぎって、組み込み SQL を備えたルーチンに対する呼び出しと CLI 呼び出しを混合することができました。組み込み SQL ルーチンは、単に既存の CLI 接続を使用します。

以上のことはこれまでどおり当てはまりますが、複雑になっている可能性があります。つまり、CLI では複数の同時接続が行えます。このことは、組み込み SQL ルーチンが呼び出されるときにどの接続を使用するかがはっきりしなくなったことを意味します。実際に、組み込みルーチンは最新のネットワーク活動に関連した接続を使用します。しかし、アプリケーションのビューから言えば、これは必ずしも決定的なものではなく、この情報を追跡することは困難です。 SQLSetConnection() は、どの接続がアクティブかをアプリケーションが*明示的に* 指定できるようにするときに使用します。アプリケーションは次に、組み込み SQL ルーチンを呼び出すことができます。

アプリケーションが CLI 呼び出しだけを使う場合、SQLSetConnection() は必要ありません。そのような場合、各ステートメント・ハンドルは暗黙で接続ハンドルに関連付けられるので、特定の CLI 関数がどの接続を適用するかで決して混乱を生じることはないからです。

戻りコード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 135. SQLSetConnection SQLSTATE

SQLSTATE	説明	解説
08003	接続がクローズされています。	提供されている接続ハンドルは、現在データベース・サーバーへのオープン接続と関連していません。
HY000	一般エラーです。	特定の SQLSTATE が用意されておらず、しかもインプリメンテーション独自の SQLSTATE も定義されていないエラーが発生しました。 SQLGetDiagRec() から引数 *MessageText* 内に戻されたエラー・メッセージに、エラーとその原因が説明されています。

制限

なし。

例

```
/* perform statements on the first connection */
cliRC = SQLSetConnection(hdbc1);

/* ... */

/* perform statements on the second connection */
cliRC = SQLSetConnection(hdbc2);
```

SQLSetConnectOption 関数 (CLI) - 接続オプションの設定

ODBC 3.0 では SQLSetConnectOption() は使用すべきでない関数なので、代わりに SQLSetConnectAttr() を使用します。

このバージョンの CLI でも引き続き SQLSetConnectOption() をサポートしていますが、最新の標準に準拠するように、SQLSetConnectAttr() を CLI プログラムで使用します。

新しい関数へのマイグレーション

注:

この使用すべきでない関数は、64 ビット環境では使用できません。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLSetConnectOptionW() です。ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

例えば、次のようなステートメントを想定します。

```
SQLSetConnectOption(
           hdbc,
           SQL_AUTOCOMMIT,
           SQL_AUTOCOMMIT_OFF);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLSetConnectAttr(
           hdbc,
           SQL_ATTR_AUTOCOMMIT,
           SQL_AUTOCOMMIT_OFF,
           0);
```

SQLSetCursorName 関数 (CLI) - カーソル名の設定

カーソル名をステートメント・ハンドルに関連付けます。

CLI はカーソル名を暗黙で生成するので、この関数はオプションです。暗黙のカーソル名を使えるようになるのは、ステートメント・ハンドル上で動的 SQL が準備済みになってからです。

仕様:

- **CLI** 1.1
- **ODBC** 1.0

- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLSetCursorNameW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN   SQLSetCursorName (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLCHAR           *CursorName,      /* szCursor */
               SQLSMALLINT       NameLength);      /* cbCursor */
```

関数引数

表 136. SQLSetCursorName 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CursorName*	入力	カーソル名
SQLSMALLINT	*NameLength*	入力	*CursorName* 引数を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数。

使用法

CLI は常に、照会を直接準備または実行するときに、内部生成のカーソル名を生成し、使用します。 SQLSetCursorName() では、アプリケーションで定義されるカーソル名を SQL ステートメント (定位置 UPDATE または DELETE) で使用できます。 CLI は、内部名にこの名前をマップします。ハンドルがドロップされるか、またはこのステートメント・ハンドルに他の SQLSetCursorName() が呼び出されるまで、その名前はステートメント・ハンドルに関連付けられたままになります。

SQLGetCursorName() はアプリケーションが設定した名前を戻しますが (名前が設定されている場合)、位置指定された UPDATE および DELETE ステートメントと関連したエラー・メッセージは、内部名を参照します。それで、位置指定された UPDATE および DELETE の場合は SQLSetCursorName() を使用しないで、SQLGetCursorName() を呼び出せば取得できる内部名を使用します。

カーソル名は、次の規則に従う必要があります。

- 接続内のすべてのカーソル名はユニークでなければなりません。
- 各カーソル名の長さは、128 バイト以下です。 128 バイトより長いカーソル名を設定しようとすると、そのカーソル名は 128 バイトに切り捨てられます。 (警告は生成されません。)
- 内部で生成される名前の先頭文字は SQLCUR または SQL_CUR なので、アプリケーションは内部名と競合しないように、先頭が SQLCUR または SQL_CUR のカーソル名を入力してはなりません。
- SQL でカーソル名は ID と見なされるため、先頭は英字 (a から z までと A から Z まで) で、その後は数字 (0 から 9 まで)、英字、または下線文字 (_) の任意の組み合わせでなければなりません。

- 上記の文字以外の文字 (各国語セットや 2 バイト文字セットの文字など) を含むカーソル名を使用できるようにするには、アプリケーションは二重引用符 (") でカーソル名を囲む必要があります。
- 入力カーソル名が二重引用符で囲まれていないと、入力カーソル名のストリングからすべての先行ブランクと後続ブランクが除去されます。

処理の効率を上げるには、 *CursorName* バッファーに先行スペースや後書きスペースを入れないようにしてください。 *CursorName* バッファーに区切り ID が入っている場合、アプリケーションは先頭の二重引用符を *CursorName* バッファーの先頭文字として配置します。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 137. SQLSetCursorName SQLSTATE

SQLSTATE	説明	解説
34000	カーソル名が無効です。	引数 *CursorName* で指定されたカーソル名は無効です。カーソル名は「SQLCUR」または「SQL_CUR」から始まっているか、または、カーソル命名規則 (先頭が a から z までか A から Z までで、その後に英字、数字、下線文字 (_) の任意の組み合わせが続く) に違反しています。 引数 *CursorName* で指定されたカーソル名はすでに存在しています。 カーソル名の長さが、 SQL_MAX_CURSOR_NAME_LEN 引数のある `SQLGetInfo()` が戻した値よりも大きいです。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
58004	予期しないシステム障害です。	回復不能システム・エラー。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	*CursorName* は、NULL ポインターでした。

表 137. SQLSetCursorName SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	ステートメント・ハンドル上にオープン・カーソルまたは位置指定されたカーソルがあります。 実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY090	ストリングまたはバッファーの長さが無効です。	引数 *NameLength* は **0** より小さく、SQL_NTS と等しくありませんでした。

許可

なし。

例

```
/* set the name of the cursor */
rc = SQLSetCursorName(hstmtSelect, (SQLCHAR *)"CURSNAME", SQL_NTS);
```

SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定

記述子レコードの単一のフィールドの値を設定します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLSetDescFieldW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLSetDescField  (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       FieldIdentifier,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        BufferLength);
```

関数引数

表 138. SQLSetDescField 引数

データ・タイプ	引数	使用法	説明
SQLHDESC	*DescriptorHandle*	入力	記述子ハンドル。
SQLSMALLINT	*RecNumber*	入力	アプリケーションが設定しようとしているフィールドを含んでいる記述子レコードを示します。記述子レコードは 0 から数え、レコード番号 0 がブックマーク・レコードになります。 *RecNumber* 引数は、ヘッダー・フィールドの場合には無視されます。
SQLSMALLINT	*FieldIdentifier*	入力	値を設定しようとしている記述子のフィールドを示します。詳細は、記述子の *FieldIdentifier* 引数の値リストを参照してください。
SQLPOINTER	*ValuePtr*	入力	記述子情報が入っているバッファーへのポインター、または 4 バイトからなる値。データ・タイプは、*FieldIdentifier* の値により異なります。 *ValuePtr* が 4 バイトからなる値である場合、*FieldIdentifier* 引数の値に応じて、その 4 バイトすべてが使われるか、あるいは 4 バイトのうち 2 つだけが使われます。
SQLINTEGER	*BufferLength*	入力	*FieldIdentifier* が ODBC で定義されたフィールドであり、かつ *ValuePtr* が文字ストリングかバイナリー・バッファーを指している場合、この引数は **ValuePtr* の長さでなければなりません。文字ストリング・データの場合、*BufferLength* の内容は、そのストリングのバイト数でなければなりません。また、*FieldIdentifier* が ODBC で定義されたフィールドであり、かつ *ValuePtr* が整数である場合には、*BufferLength* は無視されます。 *FieldIdentifier* がドライバーで定義されたフィールドである場合には、アプリケーションは *BufferLength* 引数を設定してそのフィールドの性質を示します。*BufferLength* には以下の値が入ります。 • *ValuePtr* が文字ストリングを指すポインターの場合、 *BufferLength* は、そのストリングを格納するのに必要なバイト数、または SQL_NTS です。 • *ValuePtr* がバイナリー・バッファーを指すポインターの場合、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *BufferLength* に入れます。それによって、*BufferLength* には負の値が入ります。 • *ValuePtr* が文字ストリングまたはバイナリー・ストリング以外の値を指すポインターの場合、*BufferLength* の値は SQL_IS_POINTER でなければなりません。 • *ValuePtr* に固定長の値が入っている場合、*BufferLength* は状況に応じて SQL_IS_INTEGER、SQL_IS_UINTEGER、SQL_IS_SMALLINT、または SQL_IS_USMALLINT のいずれかとなります。

使用法

アプリケーションは SQLSetDescField() を呼び出すことにより、任意の記述子フィールドを一度に 1 つずつ設定できます。 SQLSetDescField() への 1 回の呼び出しで、単一の記述子にある単一のフィールドを設定します。その対象のフィールドが設定可能なものであれば、この関数を呼び出して、任意の記述子タイプの任意のフィールドを設定できます。詳しくは、記述子ヘッダーとレコード・フィールド初期設定の値を参照してください。

注: SQLSetDescField() への呼び出しが失敗した場合、 *RecNumber* 引数に示された記述子レコードの内容は定義されません。

この関数の 1 回の呼び出しで、他の関数を呼び出して複数の記述子フィールドを設定することができます。 SQLSetDescRec() 関数は、列やパラメーターにバインドされたデータ・タイプおよびバッファーに影響する様々なフィールド (SQL_DESC_TYPE、 SQL_DESC_DATETIME_INTERVAL_CODE、 SQL_DESC_OCTET_LENGTH、 SQL_DESC_PRECISION、 SQL_DESC_SCALE、 SQL_DESC_DATA_PTR、 SQL_DESC_OCTET_LENGTH_PTR、および SQL_DESC_INDICATOR_PTR フィールド) を設定します。また、SQLBindCol() や SQLBindParameter() を使用すれば、列やパラメーターをバインドするための完全な仕様を作成できます。これらの関数はそれぞれ、1 回の関数呼び出しで、特定のグループの記述子フィールドを設定します。

SQLSetDescField() を呼び出してから、バインド用ポインター (SQL_DESC_DATA_PTR、SQL_DESC_INDICATOR_PTR、または SQL_DESC_OCTET_LENGTH_PTR) にオフセットを追加すれば、バインド用バッファーを変更することができます。こうすれば、SQLBindCol() や SQLBindParameter() を呼び出さなくてもバインド用バッファーが変更されます。これでアプリケーションは、例えば SQL_DESC_DATA_TYPE などの他のフィールドを変更するという手間をかけずに済むので、SQL_DESC_DATA_PTR を短時間で変更できるようになります。

記述子のヘッダー・フィールドの設定は、 *RecNumber* を 0 にして SQLSetDescField() を呼び出し、さらに適切な *FieldIdentifier* を呼び出して行います。多くのヘッダー・フィールドにはステートメント属性が入っていますが、それらも SQLSetStmtAttr() への呼び出しで設定できます。これにより、アプリケーションは最初に記述子ハンドルを取得することなく、ステートメント属性を設定することが可能です。 *RecNumber* の 0 は、ブックマーク・フィールドの設定にも使用します。

注: ステートメント属性 SQL_ATTR_USE_BOOKMARKS は常に、SQLSetDescField() を呼び出してブックマーク・フィールドを設定する前に設定しなければなりません。これは必須ではありませんが、強くお勧めします。

記述子フィールドの設定順序

SQLSetDescField() を呼び出して記述子フィールドを設定する場合、アプリケーションは以下に示す特定の順序に従う必要があります。

- アプリケーションはまず最初に SQL_DESC_TYPE、SQL_DESC_CONCISE_TYPE、または SQL_DESC_DATETIME_INTERVAL_CODE フィールドを設定しなければなりません。

  注: SQL_DESC_DATETIME_INTERVAL_CODE は、ODBC では定義されていますが、CLI ではサポートされていません。
- これらのフィールドのいずれかを設定したなら、アプリケーションはデータ・タイプの属性を設定することができ、ドライバーはデータ・タイプの属性フィールドをそのデータ・タイプの適切なデフォルト値に設定します。タイプ属性フィールドのデフォルト値が自動的に設定されることにより、アプリケーションがデータ・タイプを指定すると、記述子が常に使用できるようになっています。アプリケーションが明示的にデータ・タイプ属性を設定すると、デフォルトの属性はオーバーライドされます。
- ステップ 1 に示されているいずれかのフィールドが設定され、データ・タイプ属性も設定された後は、アプリケーションは SQL_DESC_DATA_PTR を設定できます。これを行うと、記述子フィールドの整合性チェックをするよう要求されます。アプリケーションが SQL_DESC_DATA_PTR フィールドの設定後にデータ・タイプや属性を変更すると、ドライバーは SQL_DESC_DATA_PTR を NULL ポインターに設定して、そのレコードをアンバインドします。こうなると、アプリケーションは適切なステップを順番どおりに実行しないと記述子レコードが使用できません。

記述子フィールドの初期設定

記述子を割り当てる時点で、その記述子内のフィールドはデフォルト値に初期設定したり、デフォルト値なしで初期設定したり、あるいは記述子のタイプを定義しないでおくことができます。詳細は、記述子ヘッダーとレコード・フィールド初期設定の値を参照してください。

IRD のフィールドにデフォルト値があるのは、ステートメントが作成または実行され、その IRD が移植された後だけであり、ステートメント・ハンドルまたは記述子が割り当てられた時点ではありません。 IRD が移植されてしまうまでは、IRD のフィールドにアクセスしようとするとエラーが返されます。

一部の記述子フィールドは、1 つ以上 (ただし全部ではない) の記述子タイプに対して定義されます (ARD と IRD、および APD と IPD)。フィールドを記述子のタイプに対して定義していないと、どの関数もその記述子を使用する必要がなくなります。記述子は実際のデータ構造ではなく、データの論理ビューであるため、これらの余分のフィールドは定義済みフィールドに対して何の影響もありません。

SQLGetDescField() がアクセスできるフィールドは、必ずしも SQLSetDescField() によって設定されるとはかぎりません。 SQLSetDescField() で設定できるフィールドは、記述子ヘッダーおよびレコード・フィールド初期設定の値のリストで説明されています。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 139. SQLSetDescField SQLSTATE

SQLSTATE	説明	解説
01000	通常の警告	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S02	オプション値が変更されました。	SQL の制約もしくは要件が原因で、 **ValuePtr* に指定された値 (*ValuePtr* がポインターだった場合) または *ValuePtr* 内の値 (*ValuePtr* が 4 バイトからなる値だった場合) を CLI がサポートしていなかったか、 **ValuePtr* が無効だったため、CLI がそれに近い値で置き換えました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
07009	記述子索引が無効です。	*FieldIdentifier* 引数はヘッダー・フィールドでしたが、 *RecNumber* 引数が 0 ではありませんでした。 *RecNumber* 引数が 0 で、*DescriptorHandle* は IPD でした。 *RecNumber* 引数は 0 未満でした。
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	*DescriptorHandle* が関連付けられていた StatementHandle で、非同期的に実行されるある関数 (この関数ではない) が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *DescriptorHandle* と関連する *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY016	インプリメンテーション行の記述子を修正できません。	*DescriptorHandle* 引数は IRD に関連付けられていましたが、*FieldIdentifier* 引数が SQL_DESC_ARRAY_STATUS_PTR ではありませんでした。
HY021	記述子情報が矛盾します。	記述子の Type フィールド、または TYPE フィールドと関連する他のフィールドは、有効でなかったか、整合性がとれていませんでした。 TYPE フィールドが有効な CLI C タイプではありませんでした。 整合性チェック時にチェックされた記述子情報は、整合性がとれていませんでした。

表 139. SQLSetDescField SQLSTATE (続き)

SQLSTATE	説明	解説
HY091	記述子フィールド ID が無効です。	*FieldIdentifier* 引数に指定された値が、 CLI の定義済みフィールドではなく、定義済み値でもありませんでした。 SQL_DESC_COUNT フィールド内の値よりも大きい値が *RecNumber* 引数に指定されました。 *FieldIdentifier* 引数が SQL_DESC_ALLOC_TYPE でした。
HY092	オプション・タイプが範囲外です。	*Attribute* 引数に指定された値が有効ではありませんでした。
HY094	位取りの値が無効です。	*pfParamType* に指定された値が SQL_DECIMAL または SQL_NUMERIC であり、 *DecimalDigits* に指定された値が 0 より小さいかまたは引数 *pcbColDef* (精度) の値より大きい値でした。 *pfParamType* に指定された値が SQL_C_TYPE_TIMESTAMP で、 *pfParamType* に指定された値が SQL_CHAR または SQL_VARCHAR のどちらかであり、 *DecimalDigits* に指定された値が 0 より小さいかまたは 9 より大きい値でした。 *pfParamType* に指定された値が SQL_C_TIMESTAMP_EXT で、 DecimalDigits に指定された値が 0 より小さいかまたは 12 より大きい値でした。
HY105	パラメーター・タイプが無効です。	SQL_DESC_PARAMETER_TYPE に指定された値が無効でした。 (詳しくは、SQLBindParameter() の『*InputOutputType* 引数』の項を参照してください。)

制限

なし。

例

```
/* set a single field of a descriptor record */
rc = SQLSetDescField(hARD,
                     1,
                     SQL_DESC_TYPE,
                     (SQLPOINTER)SQL_SMALLINT,
                     SQL_IS_SMALLINT);
```

SQLSetDescRec 関数 (CLI) - 列またはパラメーター・データ用の複数の記述子フィールドの設定

SQLSetDescRec() 関数は、列またはパラメーター・データにバインドされているデータ・タイプとバッファーに影響を与える複数の記述子フィールドを設定します。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

構文

```
SQLRETURN    SQLSetDescRec    (SQLHDESC          DescriptorHandle,
                               SQLSMALLINT       RecNumber,
                               SQLSMALLINT       Type,
                               SQLSMALLINT       SubType,
                               SQLLEN            Length,
                               SQLSMALLINT       Precision,
                               SQLSMALLINT       Scale,
                               SQLPOINTER        DataPtr,
                               SQLLEN            *StringLengthPtr,
                               SQLLEN            *IndicatorPtr);
```

関数引数

表 140. SQLSetDescRec 引数

データ・タイプ	引数	使用法	説明
SQLHDESC	*DescriptorHandle*	入力	記述子ハンドル。 IRD ハンドルであってはなりません。
SQLSMALLINT	*RecNumber*	入力	設定するフィールドを入れる記述子レコードを示します。記述子レコードは 0 から数え、レコード番号 0 がブックマーク・レコードになります。この引数は、0 以上でなければなりません。 *RecNumber* が SQL_DESC_COUNT 値より大きい場合、SQL_DESC_COUNT は *RecNumber* の値に変更されます。
SQLSMALLINT	*Type*	入力	記述子レコードに SQL_DESC_TYPE フィールドを設定する値。
SQLSMALLINT	*SubType*	入力	SQL_DATETIME タイプのレコードの場合は、この値が SQL_DESC_DATETIME_INTERVAL_CODE フィールドを設定する値になります。
SQLLEN	*Length*	入力	記述子レコードに SQL_DESC_OCTET_LENGTH フィールドを設定する値。
SQLSMALLINT	*Precision*	入力	記述子レコードに SQL_DESC_PRECISION フィールドを設定する値。
SQLSMALLINT	*Scale*	入力	記述子レコードに SQL_DESC_SCALE フィールドを設定する値。
SQLPOINTER	*DataPtr*	据え置き入出力	記述子レコードに SQL_DESC_DATA_PTR フィールドを設定する値。 *DataPtr* を NULL ポインターに設定して、 SQL_DESC_DATA_PTR フィールドを NULL ポインターに設定することができます。
SQLLEN *	*StringLengthPtr*	据え置き入出力	記述子レコードに SQL_DESC_OCTET_LENGTH_PTR フィールドを設定する値。 *StringLengthPtr* を NULL ポインターに設定して、SQL_DESC_OCTET_LENGTH_PTR フィールドを NULL ポインターに設定することができます。
SQLLEN *	*IndicatorPtr*	据え置き入出力	記述子レコードに SQL_DESC_INDICATOR_PTR フィールドを設定する値。 *IndicatorPtr* を NULL ポインターに設定して、 SQL_DESC_INDICATOR_PTR フィールドを NULL ポインターに設定することができます。

使用法

アプリケーションは、SQLSetDescRec() を呼び出して、単一の列またはパラメーターに以下のフィールドを設定できます。

- SQL_DESC_TYPE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_DATA_PTR
- SQL_DESC_OCTET_LENGTH_PTR
- SQL_DESC_INDICATOR_PTR

SQL_DESC_DATETIME_INTERVAL_CODE を更新できるのは、 SQL_DESC_TYPE が SQL_DATETIME を指示している場合だけです。

注: SQLSetDescRec() への呼び出しが失敗した場合、 *RecNumber* 引数で識別される記述子レコードの内容は未定義です。

列またはパラメーターをバインドする際、 SQLSetDescRec() を使うと、SQLBindCol() や SQLBindParameter() を呼び出したり、 SQLSetDescField() を何回も呼び出したりしないで、バインド処理に影響を与える複数のフィールドを変更することができます。 SQLSetDescRec() を使うと、現在ステートメントと関連していない記述子にフィールドを設定できます。 SQLBindParameter() を使用すると、 1 回の呼び出しで APD と IPD の両方に設定できるフィールド数が SQLSetDescRec() よりも多く、しかも記述子ハンドルも必要ないことに注目してください。

ステートメント属性 SQL_ATTR_USE_BOOKMARKS は、必ず、 *RecNumber* 引数を 0 にしてブックマーク・フィールドを設定し、 SQLSetDescRec() を呼び出す前に設定してください。これは必須ではありませんが、強くお勧めします。

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 141. SQLSetDescRec SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)

表 141. SQLSetDescRec SQLSTATE (続き)

SQLSTATE	説明	解説
07009	記述子索引が無効です。	*RecNumber* 引数は 0 に設定されており、 *DescriptorHandle* は IPD ハンドルでした。 *RecNumber* 引数は 0 未満でした。 *RecNumber* 引数はデータ・ソースがサポートできる列やパラメーターの最大数より大きな値になっており、 *DescriptorHandle* 引数は APD、IPD、または ARD でした。 *RecNumber* 引数は 0 と等しく、 *DescriptorHandle* 引数は暗黙的に割り当てられた APD を参照していました。 (このエラーは、明示的に割り当てられたアプリケーション記述子が APD か ARD かは実行時までは分からないので、明示的に割り当てられたアプリケーション記述子では発生しません。)
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 `SQLGetDiagRec()` から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY010	関数のシーケンス・エラーです。	*DescriptorHandle* が関連付けられていた *StatementHandle* で、非同期的に実行されるある関数 (この関数ではない) が呼び出され、この関数が呼び出された時点でまだ実行中でした。 *DescriptorHandle* と関連する *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーター用のデータの送信前に、この関数が呼び出されました。
HY013	予期しない、メモリーのハンドル・エラーです。	DB2 CLI は、この関数の実行または完了をサポートするために必要なメモリーにアクセスできませんでした。
HY016	インプリメンテーション行の記述子を修正できません。	*DescriptorHandle* 引数は、IRD と関連していました。
HY021	記述子情報が矛盾します。	記述子の *Type* フィールド、または TYPE フィールドと関連する他のフィールドは、有効でなかったか、整合性がとれていませんでした。 整合性チェック時にチェックされた記述子情報は、整合性がとれていませんでした。

表 141. SQLSetDescRec SQLSTATE (続き)

SQLSTATE	説明	解説
HY094	位取りの値が無効です。	*pfParamType* に指定された値が SQL_DECIMAL または SQL_NUMERIC であり、 *DecimalDigits* に指定された値が 0 より小さいかまたは引数 *pcbColDef* (精度) の値より大きい値でした。 *pfParamType* に指定された値が SQL_C_TYPE_TIMESTAMP で、*pfParamType* に指定された値が SQL_CHAR または SQL_VARCHAR のどちらかであり、 *DecimalDigits* に指定された値が 0 より小さいかまたは 9 より大きい値でした。 *pfParamType* に指定された値が SQL_C_TIMESTAMP_EXT で、DecimalDigits に指定された値が 0 より小さいかまたは 12 より大きい値でした。

制限

なし。

例

```
SQLSMALLINT type;
SQLINTEGER length, datalen;
SQLSMALLINT id_no;
/* ... */

/* set multiple descriptor fields for a column or parameter data */
rc = SQLSetDescRec(hARD, 1, type, 0, length, 0, 0, &id_no, &datalen, NULL);
```

SQLSetEnvAttr 関数 (CLI) - 環境属性の設定

SQLSetEnvAttr() は、現行環境の環境属性を設定します。

仕様:

- **CLI** 2.1
- **ISO CLI**

構文

```
SQLRETURN   SQLSetEnvAttr    (SQLHENV           EnvironmentHandle, /* henv */
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,          /* Value */
                              SQLINTEGER        StringLength);
```

関数引数

表 142. SQLSetEnvAttr 引数

データ・タイプ	引数	使用法	説明
SQLHENV	*EnvironmentHandle*	入力	環境ハンドル。
SQLINTEGER	*Attribute*	入力	設定する環境属性。詳細は、CLI 環境属性のリストを参照してください。
SQLPOINTER	*ValuePtr*	入力	*Attribute* の値。

表 142. SQLSetEnvAttr 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER	*StringLength*	入力	属性値が文字ストリングの場合は、バイト単位の *ValuePtr* の長さ。 *Attribute* にストリングを指示しないと、CLI は *StringLength* を無視します。

使用法

いったん設定されると、属性の値はこの環境でのすべての接続に影響します。

アプリケーションは、SQLGetEnvAttr() を呼び出すことで、現行の属性値を取得することができます。

SQLSetEnvAttr() を使って設定できる属性の詳細は、CLI 環境属性のリスト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 143. SQLSetEnvAttr SQLSTATE

SQLSTATE	説明	解説
HY011	この時点で無効な操作です。	環境ハンドルに関する接続ハンドルが割り振られている間は、アプリケーションは環境属性を設定できません。
HY024	無効な属性値。	指定されている *Attribute* 値に対して、**ValuePtr* に無効値を指定しました。
HY090	ストリングまたはバッファーの長さが無効です。	*StringLength* 引数は 0 より小さい値でしたが、SQL_NTS ではありませんでした。
HY092	オプション・タイプが範囲外です。	無効な *Attribute* 値を指定しました。
HYC00	ドライバーが使用できません。	指定した *Attribute* は、CLI ではサポートされません。 指定済みの *Attribute* 値が与えられているので、引数 *ValuePtr* に指定した値はサポートされません。

制限

なし。

例

```
/* set environment attribute */
cliRC = SQLSetEnvAttr(henv, SQL_ATTR_OUTPUT_NTS, (SQLPOINTER) SQL_TRUE, 0);
```

SQLSetParam 関数 (CLI) - バッファーまたは LOB ロケーターへの 1 つのパラメーター・マーカーのバインド

ODBC 2.0 以降では SQLSetParam() は使用すべきでない関数なので、代わりに SQLBindParameter() を使用します。

このバージョンの CLI でも引き続き SQLSetParam() をサポートしていますが、最新の標準に準拠するように、SQLBindParameter() を CLI プログラムで使用します。

新しい関数へのマイグレーション

CLI 関数 SQLBindParameter() は機能的には関数 SQLSetParam() と同じです。どちらも似通った引数値とタイプをとり、同じ動作を示し、同じ戻りコードを戻します。相違点としては SQLSetParam() は、パラメーター・タイプとバッファーの最大長を指定する *InputOutputType* または *BufferLength* 引数を持ちません。SQLSetParam() を呼び出すことは、*InputOutputType* 引数を SQL_PARAM_INPUT に設定し、 *BufferLength* 引数を SQL_SETPARAM_VALUE_MAX に設定したうえで、SQLBindParameter() を呼び出すことと同等です。

例えば、次のようなステートメントを想定します。

```
SQLSetParam(hstmt, 1, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
            &parameter1, NULL);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

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

SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定

行セット内のカーソル位置を設定します。

仕様:

- **CLI** 5.0
- **ODBC** 1

構文

```
SQLRETURN   SQLSetPos (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLSETPOSIROW     RowNumber,        /* irow */
               SQLUSMALLINT      Operation,        /* fOption */
               SQLUSMALLINT      LockType);        /* fLock */
```

関数引数

表 144. SQLSetPos 引数

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

表 144. SQLSetPos 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLSETPOSIROW	*RowNumber*	入力	*Operation* 引数の指定操作を実行する行セット内の行位置。 *RowNumber* が 0 であると、操作は行セットの各行に適用されます。 追加情報については、RowNumber 引数を参照してください。
SQLUSMALLINT	*Operation*	入力	以下を実行する操作です。 • SQL_POSITION • SQL_REFRESH • SQL_UPDATE • SQL_DELETE • SQL_ADD ODBC は以下の操作も指定しますが、これは後方互換性を保つために過ぎません。この操作は CLI でもサポートされています。 • SQL_ADD CLI は、SQLSetPos() 呼び出しで SQL_ADD をサポートしていますが、 *Operation* 引数を SQL_ADD に設定して `SQLBulkOperations()` を使用するようお勧めします。
SQLUSMALLINT	*LockType*	入力	*Operation* 引数での指定操作を実行後、行のロック方法を指定します。 • SQL_LOCK_NO_CHANGE ODBC は以下の操作も指定しますが、CLI ではサポートされません。 • SQL_LOCK_EXCLUSIVE • SQL_LOCK_UNLOCK 追加情報については、LockType 引数を参照してください。

使用法

RowNumber 引数

RowNumber 引数は行セットの行数を指定し、それに対して *Operation* 引数の指定操作を実行します。 *RowNumber* が 0 であると、操作は行セットの各行に適用されます。 *RowNumber* は、0 以上、行セット内の行数以下にする必要があります。

注 C 言語では、配列は 0 ベースで、*RowNumber* 引数は 1 ベースです。例えば、行セットの第 5 行を更新する場合、アプリケーションは行セット・バッファーを配列指標 4 で変更しますが、 *RowNumber* には 5 を指定します。

すべての操作で、カーソルは *RowNumber* によって指定される行に置かれます。以下の操作にはカーソル位置が必要です。
- 位置指定の更新および削除ステートメント
- `SQLGetData()` の呼び出し

- SQL_DELETE、SQL_REFRESH、および SQL_UPDATE オプションによる SQLSetPos() の呼び出し

アプリケーションは、SQLSetPos() の呼び出し時にカーソル位置を指定することができます。一般には、SQL_POSITION または SQL_REFRESH 操作で SQLSetPos() を呼び出してカーソル位置を決めてから、位置指定の更新または削除ステートメントを実行したり、 SQLGetData() を呼び出したりします。

Operation 引数

データ・ソースがどのオプションをサポートしているのかを判別するために、アプリケーションは、カーソルのタイプに基づいて、以下のいずれかの情報タイプを使用して SQLGetInfo() を呼び出します。

- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES1

SQL_POSITION

CLI は、*RowNumber* で指定された行にカーソルを置きます。

SQL_ATTR_ROW_OPERATION_PTR ステートメント属性が示す行状況配列の内容は、 SQL_POSITION *Operation* では無視されます。

SQL_REFRESH

CLI は、*RowNumber* で指定された行にカーソルを置き、その行の行セット・バッファーのデータをリフレッシュします。 CLI がどのように行セット・バッファーのデータを戻すかについて詳しくは、行方向のバインドおよび列方向のバインドの説明の項を参照してください。

SQL_REFRESH の *Operation* を指定した SQLSetPos() は、現行のフェッチ行セット内の行の状況と内容だけを更新します。これにはブックマークのリフレッシュも含まれます。バッファー内のデータがリフレッシュされますが、再取り出しはされないため、行セットのメンバーシップは固定です。

SQLSetPos() でのリフレッシュが正常に実行されても、SQL_ROW_DELETED の行状況は変更されません。行セット内の削除された行は、次のフェッチが行われるまでは削除されたものとしてマークされたままです。カーソルがパッキングをサポートしている場合、それらの行は次のフェッチで完全になくなります (この場合、削除された行は後続の SQLFetch() または SQLFetchScroll() では、戻されません)。

SQLSetPos() でのリフレッシュが正常に実行されると、 SQL_ROW_ADDED の行状況が SQL_ROW_SUCCESS に変更されます (行状況配列がある場合)。

SQLSetPos() でのリフレッシュにより、 SQL_ROW_UPDATED の行状況が行の新しい状況に変更されます (行状況配列がある場合)。

行での SQLSetPos() 操作でエラーが発生すると、行状況は SQL_ROW_ERROR に設定されます (行状況配列がある場合)。

SQL_CONCUR_ROWVER または SQL_CONCUR_VALUES の SQL_ATTR_CONCURRENCY ステートメント属性でオープンするカーソルの場合、 SQLSetPos() でのリフレッシュによって、データ・ソースが使用する楽観並行値を更新し、行の変更を検出します。これは、リフレッシュされる各行ごとに生じます。

SQL_REFRESH *Operation* では、行状況配列の内容が無視されます。

SQL_UPDATE

CLI は、*RowNumber* で指定された行にカーソルを置き、行セット・バッファー内の値 (SQLBindCol() の中の *TargetValuePtr* 引数) で、基礎となるデータ行を更新します。データの長さは、長さ/標識バッファー (SQLBindCol() の中の *StrLen_or_IndPtr* 引数) から検索されます。長さが SQL_COLUMN_IGNORE の列があれば、その列は更新されません。行の更新後、行状況配列の対応するエレメントが、 SQL_ROW_UPDATED または SQL_ROW_SUCCESS_WITH_INFO に更新されます (行状況配列がある場合)。

SQL_ATTR_ROW_OPERATION_PTR ステートメント属性が指し示す行操作配列を使用して、バルク更新中は現行行セットの行を無視するよう指示することができます。詳しくは、状況および操作配列を参照してください。

SQL_DELETE

CLI は、*RowNumber* で指定された行にカーソルを置き、基礎となるデータ行を削除します。そして、行状況配列の対応するエレメントを SQL_ROW_DELETED に変更します。行が削除された後、以下の操作はこの行にとって有効でなくなります。

- 位置指定の更新および削除ステートメント
- SQLGetData() の呼び出し
- *Operation* を SQL_POSITION 以外のものに設定して行う、 SQLSetPos() の呼び出し

削除された行は、静的およびキー・セットによって操作されるカーソルにとってはまだ可視です。しかし、(SQL_ATTR_ROW_STATUS_PTR ステートメント属性が指す) インプリメンテーション行状況配列の中の削除された行の項目は、 SQL_ROW_DELETED に変更されます。

SQL_ATTR_ROW_OPERATION_PTR ステートメント属性が指し示す行操作配列を使用して、バルク削除中は現行行セットの行を無視するよう指示することができます。詳しくは、状況および操作配列を参照してください。

SQL_ADD

ODBC は SQL_ADD *Operation* も指定しますが、これは後方互換性を保つために過ぎません。この操作は CLI でもサポートされています。しかし、*Operation* 引数を SQL_ADD に設定して、 SQLBulkOperations() を使用するようお勧めします。

LockType 引数

LockType 引数により、アプリケーションは並行性を制御することができます。一般に、並行性レベルおよびトランザクションをサポートするデータ・ソースは、*LockType* 引数の SQL_LOCK_NO_CHANGE 値だけをサポートします。

LockType 引数は、単一のステートメントで指定しますが、ロックは、同じ特権をその接続でのすべてのステートメントに与えます。特に、ある接続のあるステートメントが得たロックは、同じ接続の別のステートメントによってアンロックできます。

ODBC は、以下の *LockType* 引数を定義します。 CLI は SQL_LOCK_NO_CHANGE をサポートします。データ・ソースがサポートしているロックを判別するために、アプリケーションは、 SQL_LOCK_TYPES 情報タイプの指定された `SQLGetInfo()` を呼び出します。

表 145. 操作値

LockType 引数	**ロック・タイプ**
SQL_LOCK_NO_CHANGE	行が、SQLSetPos() の呼び出し前と同じロック状態またはアンロック状態にあるかどうかを確認します。 *LockType* のこの値によって、明示的行レベル・ロックをサポートしていないデータ・ソースでも、現行の並行性およびトランザクション分離レベルが何らかのロックを必要とする場合に、それを使用できるようになります。
SQL_LOCK_EXCLUSIVE	CLI ではサポートされない。行を排他的にロックする。
SQL_LOCK_UNLOCK	CLI ではサポートされない。行をアンロックする。

状況および操作配列

以下の状況および操作配列は、 SQLSetPos() の呼び出し時に使用します。

- 行状況配列 (SQL_DESC_ARRAY_STATUS_PTR によって IRD および SQL_ATTR_ROW_STATUS_ARRAY ステートメント属性が示される) には、行セットのデータの行ごとの状況値が含まれています。状況値は、`SQLFetch()`、`SQLFetchScroll()`、または `SQLSetPos()` の呼び出し後に、この配列に設定されます。この配列は、SQL_ATTR_ROW_STATUS_PTR ステートメント属性によって示されます。
- 行操作配列 (ARD および SQL_ATTR_ROW_OPERATION_ARRAY ステートメント属性の SQL_DESC_ARRAY_STATUS_PTR フィールドによって示される) には、 SQLSetPos() へのバルク操作呼び出しが実行されるか無視されるかを示した行セットの行ごとの値が含まれています。配列の各エレメントは、SQL_ROW_PROCEED (デフォルト値) または SQL_ROW_IGNORE に設定されます。この配列は、SQL_ATTR_ROW_OPERATION_PTR ステートメント属性によって示されます。

状況および操作配列のエレメント数は、行セットの行数に等しくなければなりません。 (SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性で定義されます。)

戻りコード

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

診断

表 146. SQLSetPos SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01004	データが切り捨てられました。	*Operation* 引数は SQL_REFRESH で、データ・タイプ SQL_C_CHAR または SQL_C_BINARY の 1 つ以上の列用に戻されるストリングやバイナリー・データは、非ブランク文字または非 NULL 文字のバイナリー・データの短縮形となりました。
01S01	行にエラーがあります。	*RowNumber* 引数は 0 で、*Operation* 引数で指定した操作の実行中に、1 つ以上の行でエラーが発生しました。 (複数行操作の全体ではなく 1 つ以上の行でエラーが発生すると、SQL_SUCCESS_WITH_INFO が戻され、また、単一行操作でエラーが発生すると、SQL_ERROR が戻されます。)
01S07	小数点以下切り捨てです。	*Operation* 引数は SQL_REFRESH で、アプリケーション・バッファーのデータ・タイプは SQL_C_CHAR や SQL_C_BINARY ではなく、1 つ以上の列用にアプリケーション・バッファーに戻されるデータは切り捨てられました。数値データ・タイプの場合、数の小数部分は切り捨てられます。時刻およびタイム・スタンプのデータ・タイプの場合、時刻の小数部分は切り捨てられます。
07006	無効な変換です。	結果セットの列のデータ値を、SQLBindCol() 呼び出しの *TargetType* で指定されたデータ・タイプに変換できませんでした。
07009	記述子索引が無効です。	引数 *Operation* は、SQL_REFRESH または SQL_UPDATE でしたが、結果セットの列数より大きい列番号で列がバインドされていたか、または列番号が 0 より小さい値でした。
21S02	派生表の次数が列リストに一致していません。	引数 *Operation* は SQL_UPDATE で、どの列も更新不能でした。理由は、どの列もバインドされていないか、読み取り専用であるか、バインド済みの長さ/標識バッファーが SQL_COLUMN_IGNORE であったためです。
22001	ストリング・データの右側が切り捨てられました。	列への文字またはバイナリー値の割り当てが、非ブランク文字 (文字の場合) または非 NULL 文字 (バイナリー数の場合) またはバイトに切り捨てられました。
22003	数値が範囲外です。	引数 *Operation* は SQL_UPDATE で、結果セットの列への数値割り当てが、数の整数部分 (小数部分ではなく) を切り捨てました。 引数 *Operation* は SQL_REFRESH で、1 つ以上のバインド済み列に数値を戻したことで、有効数字を失った可能性があります。
22007	無効な日付時刻形式です。	引数 *Operation* は SQL_UPDATE であり、結果セットの列への日付またはタイム・スタンプ値の代入において年、月、または日フィールドが範囲外になりました。 引数 *Operation* は SQL_REFRESH であり、1 つ以上のバインド済み列用の日付またはタイム・スタンプ値の戻りで、年、月、または日フィールドが範囲外になった可能性があります。

表 146. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
22008	日時フィールドがオーバーフローしました。	*Operation* 引数は SQL_UPDATE であり、結果セットの列に送られるデータに対して日時演算を実行した結果、日時フィールド (年、月、日、時、分、または秒フィールド) がフィールドの値の許容範囲を超えたか、または、グレゴリオ暦に基づく日時の法則に対して無効な値になりました。 *Operation* 引数は SQL_REFRESH であり、結果セットから検索されるデータに対して日時演算を実行した結果、日時フィールド (年、月、日、時、分、または秒フィールド) がフィールドの値の許容範囲を超えたか、または、グレゴリオ暦に基づく日時の法則に対して無効な値になりました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY010	関数のシーケンス・エラーです。	指定した *StatementHandle* が実行状態にありませんでした。最初に SQLExecDirect()、SQLExecute()、またはカタログ関数を呼び出さずに、この関数を呼び出しました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 SQLExecute()、SQLExecDirect()、または SQLSetPos() が *StatementHandle* で呼び出され、SQL_NEED_DATA を戻しました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。 SQLFetchScroll() の呼び出し前または SQLFetch() の呼び出し後の、しかも SQL_CLOSE オプションを指定した SQLFreeStmt() の呼び出し前に、 ODBC 2.0 アプリケーションは、*StatementHandle* で SQLSetPos() を呼び出しました。
HY011	この時点で無効な操作です。	ODBC 2.0 アプリケーションが SQL_ATTR_ROW_STATUS_PTR ステートメント属性を設定しましたが、 SQLFetch()、SQLFetchScroll()、または SQLExtendedFetch() の呼び出し前に SQLSetPos() が呼び出されました。

表 146. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
HY090	ストリングまたはバッファーの長さが無効です。	*Operation* 引数は SQL_ADD、SQL_UPDATE、または SQL_UPDATE_BY_BOOKMARK であり、データ値は NULL ポインターであり、列長値は 0、 SQL_DATA_AT_EXEC、 SQL_COLUMN_IGNORE、 SQL_NULL_DATA のいずれでもでないか、または SQL_LEN_DATA_AT_EXEC_OFFSET 以下でした。 *Operation* 引数は SQL_ADD、SQL_UPDATE、または SQL_UPDATE_BY_BOOKMARK であり、データ値は NULL ポインターではなく、列長値は 0 よりも小さいが、 SQL_DATA_AT_EXEC、 SQL_COLUMN_IGNORE、 SQL_NTS、または SQL_NULL_DATA ではないか、または SQL_LEN_DATA_AT_EXEC_OFFSET 以下でした。 長さ/標識バッファーの値は SQL_DATA_AT_EXEC でした。 SQL タイプは、SQL_LONGVARCHAR、 SQL_LONGVARBINARY、または他のデータ・ソース固有のデータ・タイプでした。また、 SQLGetInfo() の情報タイプ SQL_NEED_LONG_DATA_LEN は Y でした。
HY092	オプション・タイプが範囲外です。	*Operation* 引数は SQL_UPDATE_BY_BOOKMARK、 SQL_DELETE_BY_BOOKMARK、または SQL_REFRESH_BY_BOOKMARK で、 SQL_ATTR_USE_BOOKMARKS ステートメント属性は SQL_UB_OFF に設定されました。
HY107	行の値が範囲外です。	引数 *RowNumber* の指定値は、行セットの行数より大きい値でした。
HY109	カーソルの位置が無効です。	*StatementHandle* に関連したカーソルは、前方向のみで定義されており、行セット内に置くことができませんでした。 SQLSetStmtAttr() の SQL_ATTR_CURSOR_TYPE 属性の説明を参照してください。 *Operation* 引数は SQL_UPDATE、SQL_DELETE、または SQL_REFRESH で、 *RowNumber* 引数で識別する行は、削除されているか、フェッチされていません。 *RowNumber* 引数は 0 で、*Operation* 引数は SQL_POSITION でした。
HYC00	ドライバーが使用できません。	CLI またはデータ・ソースは、 *Operation* 引数または *LockType* 引数で要求した操作をサポートしていません。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、照会タイムアウト期間が満了しました。タイムアウト期間は、 SQL_ATTR_QUERY_TIMEOUT の *Attribute* を指定した SQLSetStmtAttr() を使って設定します。

制限

なし。

例

```
/* set the cursor position in a rowset */
cliRC = SQLSetPos(hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
```

SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定

ステートメントに関連したオプションを設定します。

特定の接続に関連したすべてのステートメントのオプションを設定するために、アプリケーションは、SQLSetConnectAttr() を呼び出すことができます。

仕様:

- **CLI** 5.0
- **ODBC** 3.0
- **ISO CLI**

使用できるすべてのステートメント属性の詳細は、CLI ステートメント属性リストを参照してください。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLSetStmtAttrW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLSetStmtAttr (
               SQLHSTMT         StatementHandle,  /* hstmt */
               SQLINTEGER       Attribute,        /* fOption */
               SQLPOINTER       ValuePtr,         /* pvParam */
               SQLINTEGER       StringLength);    /* fStrLen */
```

関数引数

表 147. SQLSetStmtAttr 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLINTEGER	*Attribute*	入力	CLI ステートメント属性リスト内に載っている設定対象のオプション。

表 147. SQLSetStmtAttr 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLPOINTER	*ValuePtr*	入力	*Attribute* と関連付けられる値を指すポインター。 *Attribute* が ODBC 定義の属性である場合、アプリケーションは、 *StringLength* 記述に説明されているとおりに *StringLength* 属性を設定して、 *ValuePtr* 内の属性値を修飾する必要があるかもしれません。 *Attribute* が CLI 属性である場合、常にアプリケーションは、 *StringLength* 記述に説明されているとおりに *StringLength* 属性を設定して、 *ValuePtr* 内の属性値を修飾する必要があります。 **注:** *Attribute* が ODBC 属性の場合、その属性によっては *ValuePtr* が符号なし整数に設定されることがあります。 *Attribute* が CLI 属性の場合、その属性によっては *ValuePtr* が符号付き整数に設定されることがあります。符号なし整数が前提となっている場合に *ValuePtr* を符号付きの負の整数に設定すると、 *ValuePtr* は、警告のないまま CLI によって符号なし長精度整数として処理される可能性があります。あるいは CLI からエラー (SQLSTATE HY024) が戻される可能性もあります。

表 147. SQLSetStmtAttr 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLINTEGER	*StringLength*	入力	*Attribute* が ODBC 属性である場合、アプリケーションは、 *StringLength* 属性を以下の値に設定して、属性を修飾する必要があるかもしれません。 • *ValuePtr* が文字ストリングまたはバイナリー・バッファーを指す場合、 *StringLength* は **ValuePtr* の長さでなければなりません。文字ストリング・データの場合、*StringLength* の内容は、そのストリングのバイト数でなければなりません。 • *ValuePtr* がポインターであっても文字ストリングまたはバイナリー・バッファーを指していない場合、 *StringLength* の値は SQL_IS_POINTER でなければなりません。 • *ValuePtr* が符号なしの整数を指す場合、*StringLength* 属性は無視されます。 *Attribute* が CLI 属性である場合、アプリケーションは、 *StringLength* 属性を以下の値に設定して、属性を修飾する必要があります。 • *ValuePtr* が文字ストリングを指すポインターの場合、*StringLength* は、そのストリングを格納するのに必要なバイト数、または SQL_NTS です。 • *ValuePtr* がバイナリー・バッファーを指すポインターの場合、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *StringLength* に入れなければなりません。それによって *StringLength* は負の値になります。 • *ValuePtr* に固定長の値が入っている場合、*StringLength* は SQL_IS_INTEGER か SQL_IS_UINTEGER (どちらか適切な方) になります。 • *ValuePtr* が、文字ストリング、バイナリー・ストリング、または固定長の値以外の値を指すポインターの場合、 *StringLength* の値は SQL_IS_POINTER でなければなりません。

使用法

ステートメントのステートメント属性は、 SQLSetStmtAttr() への別の呼び出しによって変更されたり、または SQLFreeHandle() の呼び出しによってそのステートメントがドロップされたりするまでは有効です。 SQL_CLOSE、SQL_UNBIND、または SQL_RESET_PARAMS オプションを指定して SQLFreeStmt() を呼び出すと、ステートメント属性はリセットされません。

**ValuePtr* に指定されている値をデータ・ソースがサポートしない場合、ステートメント属性によっては、類似の値への置換がサポートされます。そのような場合に、CLI は SQL_SUCCESS_WITH_INFO および SQLSTATE 01S02 (オプション値が変更された) を戻します。例えば、CLI は純キー・セット・カーソルをサポートしま

す。それゆえ、CLI ではアプリケーションは、SQL_ATTR_KEYSET_SIZE 属性のデフォルト値を変更できません。すなわち CLI は、**ValuePtr* 引数で指定された他のすべての値を SQL_KEYSET_SIZE_DEFAULT に置き換えてから、SQL_SUCCESS_WITH_INFO を戻します。アプリケーションは SQLGetStmtAttr() を呼び出して、置き換えられた値を判別します。

ValuePtr によって設定した情報のフォーマットは、*Attribute* の指定により異なります。 SQLSetStmtAttr() が受け入れる属性情報のフォーマットは、 NULL で終了する文字ストリングまたは 32 ビット整数値のいずれかです。 SQLGetStmtAttr() に戻される情報のフォーマットは、SQLSetStmtAttr() での指定内容を反映したものです。例えば、SQLSetStmtAttr() の *ValuePtr* 引数が指す文字ストリングの長さは、*StringLength* バイトになりますが、それは、 SQLGetStmtAttr() から戻されるはずであった値です。

記述子の設定によるステートメント属性の設定

多くのステートメント属性は、1 つ以上の記述子のヘッダー・フィールドにも対応しています。このような属性は、SQLSetStmtAttr() の呼び出しだけでなく、SQLSetDescField() の呼び出しによっても設定可能です。 SQLSetDescField() ではなく SQLSetStmtAttr() を呼び出すことによってこれらのオプションを設定した方が、記述子ハンドルをフェッチする必要がないので便利です。

注: 1 つのステートメントで SQLSetStmtAttr() 呼び出しを行うと、他のステートメントにも影響します。それが生じるのは、ステートメントに関連したアプリケーション・パラメーター記述子 (APD) またはアプリケーション行記述子 (ARD) が明示的に割り当てられていて、しかもそれが他のステートメントにも関連している場合です。 SQLSetStmtAttr() は APD や ARD を変更するので、そのような変更は、この記述子が関連しているすべてのステートメントに適用されます。このような適用が不要な場合、アプリケーションでこの記述子と他のステートメントとの関連付けをなくして (SQLSetStmtAttr() を呼び出して、SQL_ATTR_APP_ROW_DESC フィールドまたは SQL_ATTR_APP_PARAM_DESC フィールドを別の記述子ハンドルに設定して) から、再度 SQLSetStmtAttr() を呼び出します。

記述子フィールドでもあるステートメント属性が、 SQLSetStmtAttr() の呼び出しによって設定される場合、ステートメントに関連した記述子の対応フィールドも設定されます。設定されるフィールドは、 *StatementHandle* 引数により識別されるステートメントに現に関連付けられている記述子だけに該当し、属性を設定しても、将来そのステートメントに関連付けられる記述子に影響が及ぶことはありません。ステートメント属性でもある記述子フィールドが、SQLSetDescField() の呼び出しによって設定される場合、対応するステートメント属性も設定されます。

ステートメント属性は、ステートメント・ハンドルがどの記述子に関連付けられているのかを判別します。ステートメントが割り当てられている場合 (SQLAllocHandle() を参照)、 4 つの記述子ハンドルが自動的に割り当てられ、そのステートメントに関連付けられます。明示的に割り当てられた記述子ハンドルは、ステートメントに関連付けることができます。これを行うには、SQL_HANDLE_DESC の *HandleType* で SQLAllocHandle() を呼び出して記述子ハンドルを割り当ててから、 SQLSetStmtAttr() を呼び出して記述子ハンドルをステートメントに関連付けます。

以下のステートメント属性は、記述子のヘッダー・フィールドに対応しています。

表 148. ステートメント属性

ステートメント属性	ヘッダー・フィールド	記述子
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 149. SQLSetStmtAttr SQLSTATE

SQLSTATE	説明	解説
01000	警告 !	通知メッセージ。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
01S02	オプション値が変更されました。	CLI は **ValuePtr* の指定値をサポートしていないか、または **ValuePtr* の指定値が SQL の制約および要件にかなっていないため、 CLI が同等の値を代用しました。 (関数は、SQL_SUCCESS_WITH_INFO を返します。)
08S01	通信リンクに障害が起きました。	関数が処理を完了する前に、CLI とその接続先データ・ソースとの間の通信リンクが失敗しました。
24000	カーソル状態が無効です。	*Attribute* が、SQL_ATTR_CONCURRENCY、SQL_ATTR_CURSOR_TYPE、SQL_ATTR_SIMULATE_CURSOR、または SQL_ATTR_USE_BOOKMARKS であり、カーソルがオープンしていました。
HY000	一般エラーです。	特定の SQLSTATE のないエラーが発生しました。 SQLGetDiagRec() から **MessageText* バッファー内に戻されたエラー・メッセージに、エラーとその原因が説明されています。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY009	引数の値が無効です。	*ValuePtr* に NULL ポインターが渡され、**ValuePtr* の値がストリング属性でした。

表 149. SQLSetStmtAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	*StatementHandle* で非同期実行関数が呼び出されましたが、この関数が呼び出された時点でまだ実行中でした。 *StatementHandle* で SQLExecute() または SQLExecDirect() が呼び出され、 SQL_NEED_DATA が戻されました。すべての実行時データ・パラメーターまたは列用のデータの送信前に、この関数が呼び出されました。
HY011	この時点で無効な操作です。	*Attribute* は、SQL_ATTR_CONCURRENCY、SQL_ATTR_CURSOR_TYPE、SQL_ATTR_SIMULATE_CURSOR、または SQL_ATTR_USE_BOOKMARKS であり、ステートメントは準備済みでした。
HY017	自動割り振りの記述子ハンドルについて無効な使用です。	*Attribute* 引数が SQL_ATTR_IMP_ROW_DESC または SQL_ATTR_IMP_PARAM_DESC でした。 *Attribute* 引数は SQL_ATTR_APP_ROW_DESC または SQL_ATTR_APP_PARAM_DESC であり、 **ValuePtr* の値は、暗黙的に割り当てられた記述子ハンドルでした。
HY024	属性の値が無効です。	指定されている *Attribute* 値に対して、**ValuePtr* に無効値を指定しました。 (CLI がこの SQLSTATE を戻すのは、SQL_ATTR_ACCESS_MODE などの離散的な値セットを受け入れる接続およびステートメント属性に対してのみです。)
HY090	ストリングまたはバッファーの長さが無効です。	*StringLength* 引数は 0 より小さい値でしたが、SQL_NTS ではありませんでした。
HY092	オプション・タイプが範囲外です。	引数 *Attribute* に指定された値が、このバージョンの CLI では無効なものでした。
HYC00	ドライバーが使用できません。	引数 *Attribute* に指定された値は、このバージョンの CLI ドライバーには有効な接続またはステートメント属性でしたが、データ・ソースではサポートされていませんでした。

制限

なし。

例

```
/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROW_ARRAY_SIZE,
                       (SQLPOINTER)ROWSET_SIZE,
                       0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROW_BIND_TYPE,
                       SQL_BIND_BY_COLUMN,
                       0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
```

```
                            SQL_ATTR_ROWS_FETCHED_PTR,
                            &rowsFetchedNb,
                            0);
  STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
```

SQLSetStmtOption 関数 (CLI) - ステートメント・オプションの設定

ODBC 3.0 では SQLSetStmtOption() は使用すべきでない関数なので、代わりに SQLSetStmtAttr() を使用します。

このバージョンの CLI でも引き続き SQLSetStmtOption() をサポートしていますが、最新の標準に準拠するように、SQLSetStmtAttr() を CLI プログラムで使用します。

新しい関数へのマイグレーション

注: この使用すべきでない関数は、64 ビット環境では使用できません。

例えば、次のようなステートメントを想定します。

```
SQLSetStmtOption(
           hstmt,
           SQL_ROWSET_SIZE,
           RowSetSize);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLSetStmtAttr(
           hstmt,
           SQL_ATTR_ROW_ARRAY_SIZE,
           (SQLPOINTER) RowSetSize,
           0);
```

SQLSpecialColumns 関数 (CLI) - 特殊な (行 ID) 列の取得

表のユニークな行の ID 情報 (主キーまたはユニーク索引など) を戻します。

情報は SQL 結果セット内に戻されますが、照会により作成された結果セットを処理するのに使用される関数と同じ関数を使用して情報を取り出すことができます。

仕様:

- **DB2 コール・レベル・インターフェース** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLSpecialColumnsW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』 を参照してください。

構文

```
SQLRETURN   SQLSpecialColumns(
              SQLHSTMT          StatementHandle,   /* hstmt */
              SQLUSMALLINT      IdentifierType,    /* fColType */
              SQLCHAR           *CatalogName,      /* szCatalogName */
              SQLSMALLINT       NameLength1,       /* cbCatalogName */
              SQLCHAR           *SchemaName,       /* szSchemaName */
              SQLSMALLINT       NameLength2,       /* cbSchemaName */
```

```
SQLCHAR          *TableName,       /* szTableName */
SQLSMALLINT      NameLength3,      /* cbTableName */
SQLUSMALLINT     Scope,            /* fScope */
SQLUSMALLINT     Nullable);        /* fNullable */
```

関数引数

表 150. SQLSpecialColumns 引数

データ・タイプ	引数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*IdentifierType*	入力	戻されるユニーク行 ID のタイプ。次のタイプのみがサポートされます。 • SQL_BEST_ROWID 指定した表のすべての行を固有に識別できる最適な列の集まりを戻します。 **注:** ODBC アプリケーションとの互換性を保つために、SQL_ROWVER も認識されますが、サポートされません。したがって、SQL_ROWVER を指定すると、空の結果が戻されます。
SQLCHAR *	*CatalogName*	入力	3 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	指定した表のスキーマ修飾子。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名。
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。

表 150. SQLSpecialColumns 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLUSMALLINT	*Scope*	入力	ユニーク行 ID が有効になるのに要する最短時間。 *Scope* は、SQL_SCOPE_CURROW、SQL_SCOPE_TRANSACTION、または SQL_SCOPE_SESSION のうちの 1 つでなければなりません。 • SQL_SCOPE_CURROW: 行 ID が確実に有効であるのは、その行にある間だけです。同じ行 ID の値を使用して後で再選択をすると、行が更新されていたり他のトランザクションによって削除されていた場合に、行を戻さないことがあります。 • SQL_SCOPE_TRANSACTION: 行 ID が確実に有効であるのは、現行トランザクションの持続期間中だけです。 • SQL_SCOPE_SESSION: 行 ID が確実に有効であるのは、接続の持続期間中だけです。 行 ID 値が確実に有効である期間は、現行のトランザクション分離レベルにより異なります
SQLUSMALLINT	*Nullable*	入力	NULL 値を入れることのできる特殊な列を戻すかどうかを判別します。 SQL_NO_NULL か SQL_NULLABLE でなければなりません。 • SQL_NO_NULLS - 戻される行 ID の列の集まりに、NULL 値を入れることができません。 • SQL_NULLABLE - 戻される行 ID の列の集まりに、NULL 値を入れられる列を含めることができます。

使用法

表の任意の行を固有に識別する方法が複数ある場合 (例えば、指定した表に複数のユニーク索引がある場合)、DB2 コール・レベル・インターフェース は内部の基準に基づいて最良 の行 ID 列の集まりを戻します。

表名に関連したスキーマ修飾子引数を指定しないと、スキーマ名は現行の接続にとって現在有効であるものにデフォルト設定されます。

表の任意の行を固有に識別できる列の集まりがない場合、空の結果セットが戻されます。

ユニークな行 ID 情報が戻される際の結果セットのフォームは、行 ID の各列が結果セット内の 1 行で表されるという形式です。 SQLSpecialColumns で戻される列は、SQLSpecialColumns() で戻され、 SCOPE がソートする結果セット内の列の順序を示します。

SQLSpecialColumns() への呼び出しは、多くの場合システム・カタログに対する複雑な、それゆえにリソースを消費する照会をマップするため、その使用は控え、呼び出しを繰り返す代わりに結果を保存しておくべきです。

SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続先の DBMS でサポートされている COLUMN_NAME 列の実際の長さを判別することができます。

新規の列が追加され、列の名前が将来のリリースで変更される場合でも、現行の列の位置は変更になりません。

注: IDS データ・サーバーは、すべての非フラグメント表に、 ROWID という名前の仮想列を持っています。IDS データ・サーバーにアクセスする場合、SQLSpecialColumns() 関数は、ROWID 列に関する情報を返します。

SQLSpecialColumns で戻される列

列 1 SCOPE (SMALLINT)
COLUMN_NAME における名前が、同じ行を指すことが保証されている期間。有効な値は、*Scope* 引数のものと同じです (行 ID の実際の有効範囲)。次の値のうちの 1 つが含まれています。

- SQL_SCOPE_CURROW
- SQL_SCOPE_TRANSACTION
- SQL_SCOPE_SESSION

それぞれの値の説明は、 331 ページの表 150 にある *Scope* を参照してください。

列 2 COLUMN_NAME (VARCHAR(128) 非 NULL)
表の主キーである (またはその一部である) 列の名前。

列 3 DATA_TYPE (SMALLINT 非 NULL)
列の SQL データ・タイプ。

列 4 TYPE_NAME (VARCHAR(128) 非 NULL)
DATA_TYPE 列の値に関連した名前の DBMS 文字ストリングでの表現。

列 5 COLUMN_SIZE (INTEGER)
DATA_TYPE 列の値が文字またはバイナリー・ストリングであることを示す場合、この列にはバイトの最大長が含まれます。それが GRAPHIC (DBCS) ストリングであれば、これはパラメーターの 2 バイト文字の個数です。

日付、時刻、タイム・スタンプ・データ・タイプの場合、これは文字に変換された場合に値を表示するために必要な SQLCHAR または SQLWCHAR エレメントの数の合計です。

数値データ・タイプの場合、これは結果セット内の NUM_PREC_RADIX 列の値に基づいて、列に許可されている総桁数または合計ビット数のいずれかです。

データ・タイプ精度の表を参照してください。

列 6 BUFFER_LENGTH (INTEGER)
SQL_C_DEFAULT が SQLBindCol()、 SQLGetData() および SQLBindParameter() 呼び出しで指定された場合に、この列からのデータを保

管するための関連 C バッファーの最大バイト。この長さには、NULL 終止符文字は含まれていません。厳密な数データ・タイプを出すには、長さとして小数部や符号も考慮されます。

データ・タイプ長の表を参照してください。

列 7 DECIMAL_DIGITS (SMALLINT)
列のスケール。スケールが適用できないデータ・タイプの場合は NULL が戻されます。データ・タイプ・スケールの表を参照してください。

列 8 PSEUDO_COLUMN (SMALLINT)
列が DB2 コール・レベル・インターフェースだけで戻される疑似列であるかどうかを示します。

- SQL_PC_NOT_PSEUDO

DB2 DBMS は、疑似列をサポートしません。 ODBC アプリケーションは、IBM 以外の RDBMS サーバーから以下の値を受け取る可能性があります。

- SQL_PC_UNKNOWN
- SQL_PC_PSEUDO

戻りコード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 151. SQLSpecialColumns SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 `SQLCancel()` がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	*TableName* が NULL です。

表 151. SQLSpecialColumns SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。 長さ引数のうちの 1 つの値は、その修飾子または名前について DBMS でサポートされる最大長を超えています。
HY097	列タイプが範囲外です。	無効な *IdentifierType* 値を指定しました。
HY098	有効範囲のタイプが範囲外です。	無効な *Scope* 値を指定しました。
HY099	ヌル・タイプが範囲外です。	無効な *Nullable* 値を指定しました。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

例

```
/* get special columns */
cliRC = SQLSpecialColumns(hstmt,
                          SQL_BEST_ROWID,
                          NULL,
                          0,
                          tbSchema,
                          SQL_NTS,
                          tbName,
                          SQL_NTS,
                          SQL_SCOPE_CURROW,
                          SQL_NULLABLE);
```

SQLStatistics 関数 (CLI) - 基本表の索引情報および統計情報の取得

SQLStatistics() 関数は、指定された表の索引情報を取り出します。

SQLStatistics() 関数は、表および表の索引に関連したカーディナリティーとページ数も戻します。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

この情報は結果セットに戻されます。これは、照会により生成された結果セットの処理で使用するものと同じ関数を用いて取り出すことができます。

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLStatisticsW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、 5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLStatistics    (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLCHAR           *CatalogName,     /* szCatalogName */
               SQLSMALLINT       NameLength1,      /* cbCatalogName */
               SQLCHAR           *SchemaName,      /* szSchemaName */
               SQLSMALLINT       NameLength2,      /* cbSchemaName */
               SQLCHAR           *TableName,       /* szTableName */
               SQLSMALLINT       NameLength3,      /* cbTableName */
               SQLUSMALLINT      Unique,           /* fUnique */
               SQLUSMALLINT      Reserved);        /* fAccuracy */
```

関数引数

表 152. SQLStatistics 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	指定した表のスキーマ修飾子。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名。
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。

表 152. SQLStatistics 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLUSMALLINT	*Unique*	入力	戻される索引情報のタイプ。 • SQL_INDEX_UNIQUE ユニーク索引だけを戻します。 • SQL_INDEX_ALL すべての索引を戻します。
SQLUSMALLINT	*Reserved*	入力	結果セットの CARDINALITY および PAGES 列に、以下の最新情報を含むかどうかを示します。 • SQL_ENSURE : この値は、今後アプリケーションが最新の統計情報を要求するとき利用するために予約されています。**新しいアプリケーションは、この値を使用してはなりません。** この値を指定する既存のアプリケーションは、SQL_QUICK と同じ結果を受け取ります。 • SQL_QUICK : サーバーで読み取りに使用できる統計が戻されます。値が現行値であることの確認は試行されません。

使用法

SQLStatistics() 関数は、次の 2 つのタイプの情報を戻します。

- 表の統計情報 (使用できる場合)。
  - 結果セットの TYPE 列を、SQL_TABLE_STAT に設定すると、表の中の行数と、その表の保管に使用されるページ数が戻されます。
  - 結果セットの TYPE 列が索引を示す場合は、索引内のユニーク値の数と、索引の保管に使用されるページ数が戻されます。
- 各索引に関する情報。各列は結果セットの 1 行で表されます。結果セットの列については、SQLStatistics で戻される列で説明されています。結果セット内の行は、NON_UNIQUE、TYPE、INDEX_QUALIFIER、INDEX_NAME、KEY_SEQ の各列の順序になります。

多くの場合に SQLStatistics() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

表名に関連したスキーマ修飾子引数を指定しないと、スキーマ名は現行の接続にとって有効である引数にデフォルト設定されます。

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() 関数を呼び出して、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SchemaName に値として *ALL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLStatistics で戻される列

列 1 TABLE_CAT (VARCHAR(128))
索引が適用される表のカタログ名。この表にカタログがない場合、この値は NULL になります。

列 2 TABLE_SCHEM (VARCHAR(128))
TABLE_NAME を含むスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表の名前。

列 4 NON_UNIQUE (SMALLINT)
索引で重複値が禁止されるかどうかを示します。
- 索引で重複値を使用できる場合、SQL_TRUE が戻されます。
- 索引値がユニークでなければならない場合、SQL_FALSE が戻されます。
- この行が SQL_TABLE_STAT (表に関する統計情報) であることが TYPE 列に示されている場合、NULL が戻されます。

列 5 INDEX_QUALIFIER (VARCHAR(128))
DROP INDEX ステートメントで索引名を修飾するために使用するストリング。ピリオド (.) と INDEX_NAME で、索引の完全指定になります。

列 6 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 です。

列 7 TYPE (SMALLINT 非 NULL)
結果セットのこの行に含まれている情報のタイプを示します。
- SQL_TABLE_STAT は、表に関する統計情報がこの行に含まれていることを示します。
- SQL_INDEX_CLUSTERED は、索引に関する情報がこの行に含まれており、索引のタイプがクラスター索引であることを示します。
- SQL_INDEX_HASHED は、索引に関する情報がこの行に含まれており、索引のタイプがハッシュ索引であることを示します。
- SQL_INDEX_OTHER は、索引に関する情報がこの行に含まれており、その索引がクラスター索引やハッシュ索引ではないことを示します。

列 8 ORDINAL_POSITION (SMALLINT)
名前が INDEX_NAME 列に示されている索引内の列の順序を表す位置。TYPE 列の値が SQL_TABLE_STAT の場合、この列について NULL 値が戻されます。

列 9 COLUMN_NAME (VARCHAR(128))
索引内の列の名前。TYPE 列の値が SQL_TABLE_STAT の場合、この列について NULL 値が戻されます。

列 10 ASC_OR_DESC (CHAR(1))
列のソート・シーケンスです (昇順の場合は「**A**」、降順の場合は「**D**」)。TYPE 列の値が SQL_TABLE_STAT である場合、NULL 値が戻されます。

列 11 CARDINALITY (INTEGER)
- TYPE 列の値が SQL_TABLE_STAT の場合、この列には表の中の行数が含まれています。
- TYPE 列の値が SQL_TABLE_STAT でない場合、この列には索引内のユニーク値の数が含まれています。
- DBMS から情報を使用できない場合、NULL 値が戻されます。

列 12 PAGES (INTEGER)
- TYPE 列の値が SQL_TABLE_STAT の場合、この列には表を保管するのに使用するページ数が含まれています。
- TYPE 列の値が SQL_TABLE_STAT でない場合、この列には索引を保管するのに使用するページ数が含まれています。
- DBMS から情報を使用できない場合、NULL 値が戻されます。

列 13 FILTER_CONDITION (VARCHAR(128))
索引がフィルター索引である場合、これはフィルターの状態です。 DB2 サーバーはフィルター索引をサポートしていないため、常に NULL が戻されます。TYPE が SQL_TABLE_STAT である場合にも、NULL が戻されます。

表の統計を含む結果セット内の行 (TYPE は SQL_TABLE_STAT に設定される) の場合、 NON_UNIQUE、 INDEX_QUALIFIER、 INDEX_NAME、 ORDINAL_POSITION、 COLUMN_NAME、および ASC_OR_DESC の列の値が NULL 設定されます。 CARDINALITY 情報か PAGES 情報かを判別できない場合、これらの列について NULL が戻されます。

注: アプリケーションは、SQLCA の SQLERRD(3) と SQLERRD(4) フィールドを調べて表に関する特定の統計を収集することができます。ただしこれらのフィールドに戻される情報の正確度は、ステートメント内でのパラメーター・マーカーや式の使用などのさまざまな要因によって決まります。その主な要因のうち、データベース統計の正確さは制御可能です。例えば、DB2 Database for Linux, UNIX, and Windows では、`RUNSTATS` コマンドの最終実行時です。したがって、多くの場合 `SQLStatistics()` で戻された統計情報のほうが、前述の SQLCA フィールドに入れられる統計情報よりも整合性と信頼性が高くなります。

戻りコード
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 153. SQLStatistics SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 `SQLCancel()` がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	*TableName* が NULL です。
HY010	関数のシーケンス・エラーです。	実行時データ (`SQLParamData()`、`SQLPutData()`) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。 名前の長さ引数のうちの 1 つの有効値は、そのデータ・ソースについてサポートされる最大値を超えました。`SQLGetInfo()` 関数を呼び出して、サポートされる最大値を取得することができます。
HY100	Uniqueness オプション・タイプが範囲外です。	無効な *Unique* 値を指定しました。
HY101	Accuracy オプション・タイプが範囲外です。	無効な *Reserved* 値を指定しました。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 `SQLSetStmtAttr()` の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

なし。

例

```
/* get index and statistics information for a base table */
cliRC = SQLStatistics(hstmt,
                      NULL,
                      0,
                      tbSchema,
                      SQL_NTS,
                      tbName,
                      SQL_NTS,
                      SQL_INDEX_UNIQUE,
                      SQL_QUICK);
```

SQLTablePrivileges 関数 (CLI) - 表に関連する特権の取得

SQLTablePrivileges() 関数は、表と各表の関連特権のリストを戻します。

この情報は SQL 結果セットで返されます。この結果セットは、照会によって生成される結果セットを処理するため使用するのと同じ関数を使用して取得することができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLTablePrivilegesW() です。 ANSI 関数から Unicode 関数へのマッピングの詳細は、5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

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

関数引数

表 154. SQLTablePrivileges 引数

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

表 154. SQLTablePrivileges 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLCHAR *	*CatalogName*	入力	3 つの部分から成る表名のカタログ修飾子。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *PKCatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための *パターン値* が入れられるバッファー。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名で結果セットを修飾するための *パターン値* を入れられるバッファー。
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。

SchemaName および *TableName* 入力引数は、検索パターンを受け入れます。

使用法

結果は、次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、そして PRIVILEGE 列の順序になっています。指定の表に複数の特権が関連している場合、各特権は個別の行として戻されます。

ここに報告されている各特権の細分性は、列レベルで当てはまる場合があります。例えば、データ・ソースによっては、表が更新可能であればその表の中の各列も更新可能な場合があります。その他のデータ・ソースの場合、アプリケーションは、個々の列の表特権が同じであるかどうかを検出するため、`SQLColumnPrivileges()` を呼び出す必要があります。

多くの場合に `SQLTablePrivileges()` 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

アプリケーションが関数を呼び出して、結果セットの戻りを制限する試行がなされない場合があります。例えば大量の表、ビュー、および別名を含むデータ・ソースの場合、このシナリオの結果セットは非常に大きくなり、検索時間が非常に長くか

かります。長い検索時間を短縮するため、SchemaList 構成キーワードを CLI 初期設定ファイルに指定できます。そうすれば、アプリケーションが SchemaName に NULL ポインターを提供した場合に結果セットを限定できます。アプリケーションが SchemaName ストリングを指定した場合にも、出力を限定するにはやはり SchemaList キーワードを使います。したがって、指定されたスキーマ名が SchemaList ストリングでないと、空の結果セットが生成されます。

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、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SchemaName に値として *ALL または *USRLIBL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。*ALL を指定すると、CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。 IBM DB2 for IBM i サーバーで *USRLIBL を指定すると、CLI はサーバー・ジョブの現行ライブラリーで検索します。他の DB2 サーバーでは、*USRLIBL は特別な意味を持たず、CLI はパターンとして *USRLIBL を使用して検索します。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL または *USRLIBL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLTablePrivileges で戻される列

列 1 TABLE_CAT (VARCHAR(128))
カタログ表の名前。この表にカタログがない場合、この値は NULL になります。

列 2 TABLE_SCHEM (VARCHAR(128))
TABLE_NAME を含むスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表の名前。

列 4 GRANTOR (VARCHAR(128))
特権を付与したユーザーの許可 ID。

列 5 GRANTEE (VARCHAR(128))
特権が付与されたユーザーの許可 ID。

列 6 PRIVILEGE (VARCHAR(128))
表の特権。この特権は、次のストリングのうちの 1 つとすることができます。
- ALTER
- CONTROL
- DELETE
- INDEX
- INSERT

- REFERENCES
- SELECT
- UPDATE

列 7 IS_GRANTABLE (VARCHAR(3))
特権を付与されたユーザーが他のユーザーに特権を付与できるかどうかを示します。

IS_GRANTABLE 値は、「YES」、「NO」、または NULL のいずれかです。

注: CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列タイプ、内容、および順序は、ODBC の SQLProcedures() 結果セットで定義されているものと同じです。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 155. SQLTablePrivileges SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。

表 155. SQLTablePrivileges SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。 名前の長さ引数のうちの 1 つの有効値は、そのデータ・ソースについてサポートされる最大値を超えました。SQLGetInfo() 関数を呼び出して、サポートされる最大値を取得することができます。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定できます。

制限

なし。

例

```
/* get privileges associated with a table */
cliRC = SQLTablePrivileges(hstmt,
                           NULL,
                           0,
                           tbSchemaPattern,
                           SQL_NTS,
                           tbNamePattern,
                           SQL_NTS);
```

SQLTables 関数 (CLI) - 表の情報の取得

SQLTables() 関数は、接続しているデータ・ソースのシステム・カタログに保管されている表名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、照会により生成された結果セットの処理で使用するものと同じ関数を用いて取り出すことができます。

仕様:

- **CLI** 2.1
- **ODBC** 1.0

Unicode 環境での同等機能: この関数は Unicode 文字セットとともに使用することもできます。これに対応する Unicode 関数は SQLTablesW() です。ANSI と Unicode の関数マッピングに関する情報は、5 ページの『Unicode 関数 (CLI)』を参照してください。

構文

```
SQLRETURN   SQLTables       (
               SQLHSTMT         StatementHandle,   /* hstmt */
               SQLCHAR          *CatalogName,      /* szCatalogName */
               SQLSMALLINT      NameLength1,       /* cbCatalogName */
               SQLCHAR          *SchemaName,       /* szSchemaName */
               SQLSMALLINT      NameLength2,       /* cbSchemaName */
               SQLCHAR          *TableName,        /* szTableName */
               SQLSMALLINT      NameLength3,       /* cbTableName */
               SQLCHAR          *TableType,        /* szTableType */
               SQLSMALLINT      NameLength4);      /* cbTableType */
```

関数引数

表 156. SQLTables 引数

データ・タイプ	引数	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 つの部分から成る表名のカタログ修飾子。パターン *値* を含められます。ターゲットの DBMS が 3 つの部分から成る名前をサポートしておらず、かつ *CatalogName* が NULL ポインターでなく、長さ 0 のストリングを指していない場合は、空の結果セットと SQL_SUCCESS が戻されます。それ以外の場合、これは、3 パート・ネーミングをサポートする DBMS の有効なフィルターです。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *CatalogName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*SchemaName*	入力	スキーマ名で結果セットを修飾するための パターン*値* が入れられるバッファー。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *SchemaName* がヌル終了ストリングの場合は SQL_NTS。
SQLCHAR *	*TableName*	入力	表名で結果セットを修飾するための パターン*値* を入れられるバッファー。
SQLSMALLINT	*NameLength3*	入力	*TableName* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableName* がヌル終了ストリングの場合は SQL_NTS。

表 156. SQLTables 引数 (続き)

データ・タイプ	引数	使用法	説明
SQLCHAR *	*TableType*	入力	表タイプで結果セットを修飾するための *値リスト* を入れられるバッファー。 値リストは、対象の表タイプに合わせて大文字の単一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、次のものがあります。 ALIAS、HIERARCHY TABLE、INOPERATIVE VIEW、NICKNAME、MATERIALIZED QUERY TABLE、SYSTEM TABLE、TABLE、TYPED TABLE、TYPED VIEW、または VIEW。 *TableType* 引数が NULL ポインターであるか、長さゼロのストリングの場合、これは、表タイプ ID に使用できるすべての ID を指定することと同等です。 SYSTEM TABLE を指定すると、システム表とシステム・ビュー (存在する場合) の両方が戻されます。
SQLSMALLINT	*NameLength4*	入力	*TableType* を格納するのに必要な SQLCHAR エレメント (またはこの関数の Unicode 版の場合は SQLWCHAR エレメント) の数、または *TableType* がヌル終了ストリングの場合は SQL_NTS。

CatalogName、*SchemaName*、および *TableName* 入力引数は、検索パターンを受け入れます。

使用法

表の情報は、各表が結果セットの 1 行で表される結果セット内に戻されます。アプリケーションは SQLTablePrivileges() 関数を呼び出して、リスト内の表で使用できるアクセスのタイプを判別することができます。ユーザーが SELECT 特権を与えられていない表を選択した場合に、アプリケーションはそれに対処できなければなりません。

スキーマのリストだけの入手をサポートするには、以下の *SchemaName* 引数用のセマンティクスを適用します。*SchemaName* が 1 つのパーセント文字 (%) を含むストリングで、*CatalogName* および *TableName* が空ストリングの場合、結果セットには、データ・ソースの有効スキーマのリストが入ります。

TableType が 1 つのパーセント文字 (%) で、 *CatalogName*、*SchemaName*、および *TableName* が空ストリングの場合、結果セットには、データ・ソースの有効表タイプのリストが入ります。 (TABLE_TYPE 列以外のすべての列には、NULL が含まれています。)

TableType が空ストリングでない場合、大文字のリストが含まれていなければなりませんが、これは、対象となるタイプに関するコンマで区切った値となります。個々の値を単一引用符で囲んだり、すべての値全体を二重引用符で囲んだりできます。例えば、"'TABLE','VIEW'" または "TABLE,VIEW" となります。データ・ソースが指定された表タイプをサポートまたは認識しない場合、そのタイプについては何も戻されません。

アプリケーションは、NULL ポインターの SQLTables() 関数を、*SchemaName*、*TableName*、および *TableType* 引数の一部または全部について呼び出すことがありますが、その場合、結果セットの戻りを制限するような試行はなされません。例えば大量の表、ビュー、および別名を含むデータ・ソースの場合、このシナリオの結果セットは非常に大きくなり、検索時間が非常に長くかかります。3 つの構成キーワード (SCHEMALIST、SYSSCHEMA、TABLETYPE) を CLI 初期設定ファイルに指定しておけば、アプリケーションが *SchemaName* と *TableType* の一方または両方に NULL ポインターを提供した場合に結果セットを限定することができます。アプリケーションが *SchemaName* ストリングを指定した場合にも、出力を限定するにはやはり SCHEMALIST キーワードを使います。したがって、指定されたスキーマ名が SCHEMALIST ストリングでないと、空の結果セットが生成されます。

SQLTables() 関数から戻された結果セットには、所定の順序で SQLTables で戻される列にリストされている列が含まれています。これらの行は TABLE_TYPE、TABLE_CAT、TABLE_SCHEM、そして TABLE_NAME 列の順序になっています。

多くの場合に SQLTables() 関数の呼び出しはシステム・カタログに対する複雑な (したがってコストのかかる) 照会にマッピングされるため、それらの呼び出しの使用回数を少なくし、呼び出しを繰り返すのではなく結果を保存するようにしてください。

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() 関数を呼び出して、接続先の DBMS でサポートされている TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別することができます。

SchemaName に値として *ALL または *USRLIBL を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。*ALL を指定すると、CLI は接続されたデータベースですべての既存のスキーマを検索します。この動作は、CLI のデフォルトであるため、*ALL を指定する必要はありません。 IBM DB2 for IBM i サーバーで *USRLIBL を指定すると、CLI はサーバー・ジョブの現行ライブラリーで検索します。他の DB2 サーバーでは、*USRLIBL は特別な意味を持たず、CLI はパターンとして *USRLIBL を使用して検索します。また、SchemaFilter IBM Data Server Driver 構成キーワードまたは Schema List CLI/ODBC 構成キーワードを *ALL または *USRLIBL に設定することもできます。

将来のリリースでは、列が新たに追加されたり、既存の列の名前が変更されたりする可能性はありますが、現行の列の位置が変更されることはありません。

SQLTables で戻される列

列 1 TABLE_CAT (VARCHAR(128))
TABLE_SCHEM の入ったカタログの名前。この表にカタログがない場合、この値は NULL になります。

列 2 TABLE_SCHEM (VARCHAR(128))
TABLE_NAME の入ったスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128))
表、ビュー、別名、またはシノニムの名前。

列 4 TABLE_TYPE (VARCHAR(128))
TABLE_NAME 列内の名前で指定されたタイプを識別します。これには、ストリング値 'ALIAS'、'HIERARCHY TABLE'、'INOPERATIVE VIEW'、'NICKNAME'、'MATERIALIZED ' QUERY TABLE'、'SYSTEM TABLE'、'TABLE'、'TYPED TABLE'、'TYPED VIEW'、または 'VIEW' を使用することができます。

列 5 REMARKS (VARCHAR(254))
表に関する記述情報。

戻りコード

- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_STILL_EXECUTING
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

診断

表 157. SQLTables SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効です。	カーソルはすでに、ステートメント・ハンドル上にオープンされています。
40003 **08**S01	通信リンクに障害が起きました。	アプリケーションとデータ・ソースとの間の通信リンクが、関数の完了する前に失敗しました。
HY001	メモリーの割り振りが失敗しました。	DB2 CLI は、この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当てられません。プロセス・レベルのメモリーがアプリケーション・プロセスに使い尽くされた可能性があります。プロセス・レベルのメモリー制限については、オペレーティング・システムの構成を調べてください。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用できるようになりました。関数が呼び出され、その実行が完了する前に、 SQLCancel() がマルチスレッド・アプリケーション内の別のスレッドから、*StatementHandle* で呼び出されました。その関数が再び *StatementHandle* で呼び出されました。
HY009	引数の値が無効です。	*TableName* が NULL です。
HY010	関数のシーケンス・エラーです。	実行時データ (SQLParamData()、SQLPutData()) 操作中に、関数が呼び出されました。 BEGIN COMPOUND と END COMPOUND の SQL 操作中に、関数が呼び出されました。 非同期で実行中の関数 (この関数ではない) が *StatementHandle* で呼び出されましたが、この関数の呼び出し時にはまだ実行中でした。 ステートメント・ハンドル上のステートメントが準備される前にこの関数が呼び出されました。

表 157. SQLTables SQLSTATE (続き)

SQLSTATE	説明	解説
HY014	これ以上ハンドルがありません。	DB2 CLI は、リソースの制約のため、ハンドルを割り当てられません。
HY090	ストリングまたはバッファーの長さが無効です。	名前の長さ引数のうちの 1 つの値は 0 より小さい値でしたが、SQL_NTS と等しくありませんでした。 名前の長さ引数のうちの 1 つの有効値は、そのデータ・ソースについてサポートされる最大値を超えました。 SQLGetInfo() 関数を呼び出して、サポートされる最大値を取得することができます。
HYT00	タイムアウトになりました。	データ・ソースが結果セットを戻す前に、タイムアウト期間が満了しました。タイムアウト期間は、 SQLSetStmtAttr() の SQL_ATTR_QUERY_TIMEOUT 属性を使用して設定することができます。

制限

なし。

例

```
/* get table information */
cliRC = SQLTables(hstmt,
                  NULL,
                  0,
                  tbSchemaPattern,
                  SQL_NTS,
                  tbNamePattern,
                  SQL_NTS,
                  NULL,
                  0);
```

SQLTransact 関数 (CLI) - トランザクション管理

ODBC 3.0 では SQLTransact() は使用すべきでない関数なので、代わりに SQLEndTran() を使用します。

このバージョンの CLI でも引き続き SQLTransact() をサポートしていますが、最新の標準に準拠するように、SQLEndTran() を CLI プログラムで使用します。

新しい関数へのマイグレーション

例えば、次のようなステートメントを想定します。

```
SQLTransact(henv, hdbc, SQL_COMMIT);
```

上記の場合、新しい関数を使用して以下のように書き換えることができます。

```
SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
```

第 2 章 CLI の戻りコードと SQLSTATE

CLI 関数を呼び出すと、戻りコードと詳細診断 (SQLSTATE、メッセージ、SQLCA) の 2 つのレベルの診断が関数によって戻されます。

個々の CLI 関数は基本診断として関数戻りコードを戻します。SQLGetDiagRec() と SQLGetDiagField() の両方で、さらに詳しい診断情報を通知します。 DBMS で診断が行われる場合は、SQLGetSQLCA() 関数は SQLCA へアクセスできるようにします。この調整によって、アプリケーションは、戻りコードに基づいて基本的な制御の流れを扱えるようになり、 SQLSTATE と SQLCA を一緒に使用して特定の障害原因を判別したり、特定のエラー処理を実行できるようになります。

SQLGetDiagRec() も SQLGetDiagField() も両方とも、次の 3 つの部分からなる情報を戻します。

- SQLSTATE
- ネイティブのエラー。データ・ソースで診断が検出されるときは、これは SQLCODE で、そうでなければこれは -99999 に設定されます。
- メッセージ・テキスト。これは SQLSTATE に関連したメッセージ・テキストです。

SQLGetSQLCA() は、特定のフィールドにアクセスするための SQLCA を戻しますが、必要な情報が SQLGetDiagRec() または SQLGetDiagField() を使用して得られない場合にのみ使用してください。

CLI 関数戻りコード

次の表に、CLI 関数の戻りコードをすべてリストします。

表 158. CLI 関数戻りコード

戻りコード	解説
SQL_SUCCESS	関数が正常に完了しました。他に利用できる 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、警告または他の情報があります。 SQLGetDiagRec() または SQLGetDiagField() を呼び出して、 SQLSTATE およびその他の通知メッセージまたは警告を受け取ってください。 SQLSTATE のクラスは '01' です。
SQL_STILL_EXECUTING	関数は非同期に実行中で、まだ完了していません。CLI ドライバーは、関数を呼び出した後アプリケーションに制御を返しましたが、その関数は実行をまだ完了していません。
SQL_NO_DATA_FOUND	関数が正常に戻りましたが、関連データが見つかりませんでした。 SQL ステートメント実行後に戻されるときは、追加情報を入手することができ、SQLGetDiagRec() または SQLGetDiagField() を呼び出してこれを得ることができます。

表 158. CLI 関数戻りコード (続き)

戻りコード	解説
SQL_NEED_DATA	アプリケーションは SQL ステートメントを実行しようとしましたが、アプリケーションが実行時に渡されるよう指示したパラメーター・データが CLI にありませんでした。
SQL_ERROR	関数は失敗しました。 SQLGetDiagRec() または SQLGetDiagField() を呼び出して、 SQLSTATE およびその他のエラー情報を受け取ってください。
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。これはプログラミング・エラーです。さらに情報はありません。

次のコード・セグメントは、データの取り出しを停止するタイミングを、関数戻りコード SQL_NO_DATA_FOUND を使用して制御する方法を示しています。

```
while (cliRC != SQL_NO_DATA_FOUND)
{
  printf("     %-8d %-14.14s ¥n", deptnumb.val, location.val);
     /* fetch next row */
  cliRC = SQLFetch(hstmt);
  STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
}
```

CLI 用の SQLSTATES

SQLSTATE は、5 文字 (バイト) の英数字ストリングで、形式は ccsss です (cc はクラスで、sss はサブクラス)。クラスが以下の SQLSTATE は、次のとおりになります。

- '01' の場合、警告です。
- 'HY' の場合、CLI または ODBC ドライバーによって生成されます。
- 'IM' の場合、ODBC Driver Manager によって生成されます。

注: バージョン 5 より前のバージョンの CLI では、 'HY' ではなく 'S1' のクラスの SQLSTATE を返していました。 CLI ドライバーを指定するには 'S1' の SQLSTATE を返します。そして、アプリケーションは環境属性 SQL_ATTR_ODBC_VERSION を値 SQL_OV_ODBC2 に設定する必要があります。

CLI の SQLSTATE には、データベース・サーバーによって返される追加の IBM 定義 SQLSTATE と、 ODBC バージョン 3 および ISO SQL/CLI 仕様では定義されていない条件に関する CLI 定義 SQLSTATE の両方が含まれています。これによって、最大量の診断情報が戻されます。また、ODBC 環境でアプリケーションを実行している場合、 ODBC 定義 SQLSTATE を受け取ることも可能です。

アプリケーション内で SQLSTATE を使用する場合、次のガイドラインに従ってください。

- SQLGetDiagRec() を呼び出す前に関数戻りコードを必ずチェックして、診断情報を使用できるかどうかを判別してください。
- ネイティブのエラー・コードよりも SQLSTATE を使用してください。

- アプリケーションの移植性を高めるためには、 ODBC バージョン 3 および ISO SQL/CLI 仕様で定義されている CLI SQLSTATE のサブセットだけに依存性を持たせ、追加情報は通知専用として戻すようにします。アプリケーションでの依存性は、特定の SQLSTATE に基づいた論理フローの決定です。

  注: SQLSTATE のクラス (先頭 2 文字) に関する依存性を作成すると効果的な場合があります。
- 診断情報を最大限活用するために、 SQLSTATE とともにテキスト・メッセージを返してください (該当すれば、テキスト・メッセージには IBM 定義 SQLSTATE も含まれます)。アプリケーションがエラーを返した関数の名前を印刷することも効果的です。
- SQLSTATE に割り振られるストリングには、CLI によって戻されるヌル終了文字のためのスペースが必ず含められるようにしてください。

utilcli.c からのコード・セグメントには、 SQLSTATE などの診断情報を検索して表示する方法が示されています。

```
void HandleDiagnosticsPrint(SQLSMALLINT htype, /* handle type identifier */
                            SQLHANDLE hndl /* handle */ )
{
  SQLCHAR message[SQL_MAX_MESSAGE_LENGTH + 1];
  SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
  SQLINTEGER sqlcode;
  SQLSMALLINT length, i;

  i = 1;

  /* get multiple field settings of diagnostic record */
  while (SQLGetDiagRec(htype,
                       hndl,
                       i,
                       sqlstate,
                       &sqlcode,
                       message,
                       SQL_MAX_MESSAGE_LENGTH + 1,
                       &length) == SQL_SUCCESS)
  {
    printf("¥n  SQLSTATE          =
    printf("  Native Error Code =
    printf("
    i++;
  }

  printf("------------------------¥n");
}
```

アプリケーションが DB2 を呼び出す方法 (生じる何らかのエラーも含めて) をより良く理解するには、 CLI/ODBC のトレース機能を使用することができます。

CLI アプリケーションでのコンパウンド SQL (CLI) の戻りコード

戻りコードは、END COMPOUND ステートメントの SQLExecute() または SQLExecDirect() への呼び出し時に生成されます。以下に ATOMIC および NOT ATOMIC コンパウンド・ステートメントの戻りコードをリストします。

ATOMIC

- SQL_SUCCESS: すべてのサブステートメントは実行され、警告やエラーはありませんでした。
- SQL_SUCCESS_WITH_INFO: すべてのサブステートメントは正常に実行されましたが、 1 つ以上の警告がありました。 SQLGetDiagRec() または SQLGetDiagField() を呼び出して、エラーまたは警告についての追加情報を調べてください。SQLGetDiagRec() または SQLGetDiagField() を処理するために使用されるハンドルは、 BEGIN COMPOUND および END COMPOUND ステートメントを処理するのに使用するのと同じハンドルでなければなりません。
- SQL_NO_DATA_FOUND: BEGIN COMPOUND および END COMPOUND ステートメントがサブステートメントなしで実行されたか、サブステートメントが行に影響を与えることはありませんでした。
- SQL_ERROR: 1 つ以上のサブステートメントが失敗し、すべてのサブステートメントがロールバックされました。

NOT ATOMIC

- SQL_SUCCESS: すべてのサブステートメントは実行され、エラーはありませんでした。
- SQL_SUCCESS_WITH_INFO: COMPOUND ステートメントは実行されましたが、1 つ以上の警告が 1 つ以上のサブステートメントによって戻されました。SQLGetDiagRec() または SQLGetDiagField() を呼び出して、エラーまたは警告についての追加情報を調べてください。SQLGetDiagRec() または SQLGetDiagField() を処理するために使用されるハンドルは、 BEGIN COMPOUND および END COMPOUND ステートメントを処理するのに使用するのと同じハンドルでなければなりません。
- SQL_NO_DATA_FOUND: BEGIN COMPOUND および END COMPOUND ステートメントがサブステートメントなしで実行されたか、サブステートメントが行に影響を与えることはありませんでした。
- SQL_ERROR: COMPOUND ステートメントは失敗しました。少なくとも 1 つのサブステートメントがエラーを戻しました。 SQLCA を調べて、失敗したステートメントを判別してください。

第 3 章 CLI/ODBC 構成キーワード (カテゴリー別)

CLI 構成キーワードのカテゴリー別のリストです。

CLI/ODBC 構成キーワードは、下記のカテゴリーに分けられます。

- 互換性の構成キーワード
- データ・ソースの構成キーワード
- データ・タイプの構成キーワード
- エンタープライズの構成キーワード
- 環境の構成キーワード
- ファイルの DSN 構成キーワード
- 最適化の構成キーワード
- サービスの構成キーワード
- 静的 SQL の構成キーワード
- トランザクションの構成キーワード

ほとんどの CLI/ODBC 構成キーワードは `db2cli.ini` 初期設定ファイル内に設定されたり、 `SQLDriverConnect()` 関数への接続ストリング内にキーワード情報を指定することによって設定されますが、**`Trusted_Connection`** キーワードは `SQLDriverConnect()` によってのみ設定されます。

互換性の構成キーワード

互換性の構成キーワードは、 DB2 の動作を定義するために使用されます。互換性の構成キーワードを設定して、他のアプリケーションと DB2 との互換性を確保できます。

- 364 ページの『AllowInterleavedGetData CLI/ODBC 構成キーワード』
- 376 ページの『CheckForFork CLI/ODBC 構成キーワード』
- 390 ページの『CursorTypes CLI/ODBC 構成キーワード』
- 397 ページの『DeferredPrepare CLI/ODBC 構成キーワード』
- 397 ページの『DescribeCall CLI/ODBC 構成キーワード』
- 400 ページの『DescribeParam CLI/ODBC 構成キーワード』
- 401 ページの『DisableKeysetCursor CLI/ODBC 構成キーワード』
- 402 ページの『DisableMultiThread CLI/ODBC 構成キーワード』
- 402 ページの『DisableUnicode CLI/ODBC 構成キーワード』
- 403 ページの『EnableNamedParameterSupport CLI/ODBC 構成キーワード』
- 409 ページの『Interrupt CLI/ODBC 構成キーワード』
- 423 ページの『OleDbReportIsLongForLongTypes CLI/ODBC 構成キーワード』
- 424 ページの『OleDbSQLColumnsSortByOrdinal CLI/ODBC 構成キーワード』
- 440 ページの『RetCatalogAsCurrServer CLI/ODBC 構成キーワード』
- 440 ページの『RetOleDbConnStr CLI/ODBC 構成キーワード』

- 471 ページの『Trusted_Connection CLI/ODBC 構成キーワード』 (SQLDriverConnect() を使用してこのキーワードを設定します)
- 456 ページの『TimestampTruncErrToWarning CLI/ODBC 構成キーワード』

データ・ソースの構成キーワード

データ・ソースの構成に関連付けられている一般キーワードです。

- 393 ページの『DBAlias CLI/ODBC 構成キーワード』
- 378 ページの『ClientEncAlg CLI/ODBC 構成キーワード』
- 426 ページの『PWD CLI/ODBC 構成キーワード』
- 473 ページの『UID CLI/ODBC 構成キーワード』

データ・タイプの構成キーワード

データ・タイプの構成キーワードは、さまざまなデータ・タイプを DB2 でどのように報告および処理するかを定義するために使用されます。

- 373 ページの『BitData CLI/ODBC 構成キーワード』
- 385 ページの『CurrentImplicitXMLParseOption CLI/ODBC 構成キーワード』
- 395 ページの『DateTimeStringFormat CLI/ODBC 構成キーワード』
- 395 ページの『DecimalFloatRoundingMode CLI/ODBC 構成キーワード』
- 404 ページの『FloatPrecRadix CLI/ODBC 構成キーワード』
- 407 ページの『Graphic CLI/ODBC 構成キーワード』
- 412 ページの『LOBMaxColumnSize CLI/ODBC 構成キーワード』
- 413 ページの『LongDataCompat CLI/ODBC 構成キーワード』
- 414 ページの『MapBigintCDefault CLI/ODBC 構成キーワード』
- 414 ページの『MapCharToWChar CLI/ODBC 構成キーワード』
- 415 ページの『MapDateCDefault CLI/ODBC 構成キーワード』
- 416 ページの『MapDateDescribe CLI/ODBC 構成キーワード』
- 416 ページの『MapDecimalFloatDescribe CLI/ODBC 構成キーワード』
- 417 ページの『MapGraphicDescribe CLI/ODBC 構成キーワード』
- 418 ページの『MapTimeCDefault CLI/ODBC 構成キーワード』
- 418 ページの『MapTimeDescribe CLI/ODBC 構成キーワード』
- 419 ページの『MapTimestampCDefault CLI/ODBC 構成キーワード』
- 420 ページの『MapTimestampDescribe CLI/ODBC 構成キーワード』
- 420 ページの『MapXMLCDefault CLI/ODBC 構成キーワード』
- 421 ページの『MapXMLDescribe CLI/ODBC 構成キーワード』
- 424 ページの『OleDbReturnCharAsWChar CLI/ODBC 構成キーワード』
- 435 ページの『PromoteLONGVARtoLOB CLI/ODBC 構成キーワード』
- 476 ページの『XMLDeclaration CLI/ODBC 構成キーワード』

エンタープライズの構成キーワード

エンタープライズの構成キーワードは、大規模なデータベースへの接続の効率を最大限まで上げるために使用されます。

- 382 ページの『ConnectNode CLI/ODBC 構成キーワード』
- 387 ページの『CurrentPackagePath CLI/ODBC 構成キーワード』
- 387 ページの『CurrentPackageSet CLI/ODBC 構成キーワード』
- 388 ページの『CurrentRefreshAge CLI/ODBC 構成キーワード』
- 388 ページの『CurrentSchema CLI/ODBC 構成キーワード』
- 388 ページの『CurrentSQLID CLI/ODBC 構成キーワード』
- 393 ページの『DBName CLI/ODBC 構成キーワード』
- 405 ページの『GranteeList CLI/ODBC 構成キーワード』
- 406 ページの『GrantorList CLI/ODBC 構成キーワード』
- 425 ページの『OnlyUseBigPackages CLI/ODBC 構成キーワード』
- 439 ページの『ReportPublicPrivileges CLI/ODBC 構成キーワード』
- 442 ページの『ReturnSynonymSchema CLI/ODBC 構成キーワード』
- 444 ページの『SchemaList CLI/ODBC 構成キーワード』
- 445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』
- 447 ページの『SQLCODEMAP CLI/ODBC 構成キーワード』
- 453 ページの『SysSchema CLI/ODBC 構成キーワード』
- 454 ページの『TableType CLI/ODBC 構成キーワード』
- 475 ページの『UseServerMsgSP CLI/ODBC 構成キーワード』

環境の構成キーワード

環境の構成キーワードは、サーバー・マシンおよびクライアント・マシン上のさまざまなファイルのロケーションのような、環境固有の設定を定義するために使用されます。

- 383 ページの『ConnectTimeout CLI/ODBC 構成キーワード』
- 384 ページの『CurrentFunctionPath CLI/ODBC 構成キーワード』
- 409 ページの『Interrupt CLI/ODBC 構成キーワード』
- 436 ページの『QueryTimeoutInterval CLI/ODBC 構成キーワード』
- 437 ページの『ReadCommonSectionOnNullConnect CLI/ODBC 構成キーワード』
- 438 ページの『ReceiveTimeout CLI/ODBC 構成キーワード』
- 456 ページの『TempDir CLI/ODBC 構成キーワード』

ファイルの DSN 構成キーワード

ファイルの DSN 構成キーワードは、ファイルの DSN 接続の TCP/IP 設定値を設定するために使用されます。

- 370 ページの『Attach CLI/ODBC 構成キーワード』
- 371 ページの『Authentication CLI/ODBC 構成キーワード』
- 373 ページの『BIDI CLI/ODBC 構成キーワード』
- 394 ページの『Database CLI/ODBC 構成キーワード』
- 407 ページの『Hostname CLI/ODBC 構成キーワード』
- 433 ページの『Port CLI/ODBC 構成キーワード』

- 435 ページの『Protocol CLI/ODBC 構成キーワード』
- 446 ページの『ServiceName CLI/ODBC 構成キーワード』
- 445 ページの『security CLI/ODBC 構成キーワード』
- 447 ページの『SSLClientLabel CLI/ODBC 構成キーワード』
- 449 ページの『SSLClientKeystoredb CLI/ODBC 構成キーワード』
- 449 ページの『SSLClientKeystoreDBPassword CLI/ODBC 構成キーワード』
- 448 ページの『SSLClientKeystash CLI/ODBC 構成キーワード』
- 455 ページの『TargetPrincipal CLI/ODBC 構成キーワード』

最適化の構成キーワード

最適化の構成キーワードは、IBM Data Server Driver for ODBC and CLI とサーバーとの間のネットワーク・フローを高速化したり、その量を削減したりするために使用されます。

- 364 ページの『AllowGetDataLOBReaccess CLI/ODBC 構成キーワード』
- 367 ページの『AppendForFetchOnly CLI/ODBC 構成キーワード』
- 366 ページの『AppUsesLOBLocator CLI/ODBC 構成キーワード』
- 374 ページの『BlockForNRows CLI/ODBC 構成キーワード』
- 374 ページの『BlockLobs CLI/ODBC 構成キーワード』
- 378 ページの『ClientBuffersUnboundLOBS CLI/ODBC 構成キーワード』
- 380 ページの『ColumnwiseMRI CLI/ODBC 構成キーワード』
- 381 ページの『ConcurrentAccessResolution CLI/ODBC 構成キーワード』
- 385 ページの『CurrentMaintainedTableTypesForOpt CLI/ODBC 構成キーワード』
- 390 ページの『DB2Degree CLI/ODBC 構成キーワード』
- 391 ページの『DB2Explain CLI/ODBC 構成キーワード』
- 392 ページの『DB2NETNamedParam CLI/ODBC 構成キーワード』
- 392 ページの『DB2Optimization CLI/ODBC 構成キーワード』
- 398 ページの『DescribeInputOnPrepare CLI/ODBC 構成キーワード』
- 399 ページの『DescribeOutputLevel CLI/ODBC 構成キーワード』
- 404 ページの『FET_BUF_SIZE CLI/ODBC 構成キーワード』
- 405 ページの『GetDataLobNoTotal CLI/ODBC 構成キーワード』
- 410 ページの『KeepDynamic CLI/ODBC 構成キーワード』
- 411 ページの『LOBCacheSize CLI/ODBC 構成キーワード』
- 412 ページの『LOBFileThreshold CLI/ODBC 構成キーワード』
- 413 ページの『LockTimeout CLI/ODBC 構成キーワード』
- 422 ページの『MaxLOBBlockSize CLI/ODBC 構成キーワード』
- 425 ページの『OptimizeForNRows CLI/ODBC 構成キーワード』
- 438 ページの『Reopt CLI/ODBC 構成キーワード』
- 442 ページの『ReturnAliases CLI/ODBC 構成キーワード』
- 447 ページの『SkipTrace CLI/ODBC 構成キーワード』
- 451 ページの『StmtConcentrator CLI/ODBC 構成キーワード』

- 452 ページの『StreamGetData CLI/ODBC 構成キーワード』
- 453 ページの『StreamPutData CLI/ODBC 構成キーワード』
- 473 ページの『Underscore CLI/ODBC 構成キーワード』

サービスの構成キーワード

サービスの構成キーワードは、CLI/ODBC 接続に伴う問題のトラブルシューティングに役立てるために使用されます。プログラマーは、サービスの構成キーワードを、CLI プログラムがどのようにサーバーへの呼び出しに変換されるのかをより深く理解するために使用することもできます。

- 366 ページの『AppendAPIName CLI/ODBC 構成キーワード』
- 368 ページの『AppendRowColToErrorMessage CLI/ODBC 構成キーワード』
- 408 ページの『IgnoreWarnings CLI/ODBC 構成キーワード』
- 408 ページの『IgnoreWarnList CLI/ODBC 構成キーワード』
- 412 ページの『LoadXAInterceptor CLI/ODBC 構成キーワード』
- 426 ページの『Patch1 CLI/ODBC 構成キーワード』
- 429 ページの『Patch2 CLI/ODBC 構成キーワード』
- 439 ページの『ReportRetryErrorsAsWarnings CLI/ODBC 構成キーワード』
- 441 ページの『RetryOnError CLI/ODBC 構成キーワード』
- 434 ページの『ProgramID CLI/ODBC 構成キーワード』
- 434 ページの『ProgramName CLI/ODBC 構成キーワード』
- 457 ページの『Trace CLI/ODBC 構成キーワード』
- 458 ページの『TraceAPIList CLI/ODBC 構成キーワード』
- 460 ページの『TraceAPIList! CLI/ODBC 構成キーワード』
- 462 ページの『TraceComm CLI/ODBC 構成キーワード』
- 463 ページの『TraceErrImmediate CLI/ODBC 構成キーワード』
- 463 ページの『TraceFileName CLI/ODBC 構成キーワード』
- 464 ページの『TraceFlush CLI/ODBC 構成キーワード』
- 465 ページの『TraceFlushOnError CLI/ODBC 構成キーワード』
- 466 ページの『TraceLocks CLI/ODBC 構成キーワード』
- 468 ページの『TracePathName CLI/ODBC 構成キーワード』
- 466 ページの『TracePIDList CLI/ODBC 構成キーワード』
- 467 ページの『TracePIDTID CLI/ODBC 構成キーワード』
- 468 ページの『TraceRefreshInterval CLI/ODBC 構成キーワード』
- 469 ページの『TraceStmtOnly CLI/ODBC 構成キーワード』
- 470 ページの『TraceTime CLI/ODBC 構成キーワード』
- 470 ページの『TraceTimestamp CLI/ODBC 構成キーワード』
- 476 ページの『WarningList CLI/ODBC 構成キーワード』

静的 SQL の構成キーワード

静的 SQL の構成キーワードは、CLI/ODBC アプリケーションで静的 SQL ステートメントを実行するときに使用されます。

- 450 ページの『StaticCapFile CLI/ODBC 構成キーワード』
- 450 ページの『StaticLogFile CLI/ODBC 構成キーワード』
- 450 ページの『StaticMode CLI/ODBC 構成キーワード』
- 451 ページの『StaticPackage CLI/ODBC 構成キーワード』

トランザクションの構成キーワード

トランザクションの構成キーワードは、アプリケーションで使用される SQL ステートメントをコントロールおよび高速化するために使用されます。

- 368 ページの『ArrayInputChain CLI/ODBC 構成キーワード』
- 369 ページの『AsyncEnable CLI/ODBC 構成キーワード』
- 372 ページの『AutoCommit CLI/ODBC 構成キーワード』
- 376 ページの『ClientAcctStr CLI/ODBC 構成キーワード』
- 377 ページの『ClientApplName CLI/ODBC 構成キーワード』
- 379 ページの『ClientUserID CLI/ODBC 構成キーワード』
- 380 ページの『ClientWrkStnName CLI/ODBC 構成キーワード』
- 381 ページの『CommitOnEOF CLI/ODBC 構成キーワード』
- 384 ページの『ConnectType CLI/ODBC 構成キーワード』
- 389 ページの『CursorHold CLI/ODBC 構成キーワード』
- 422 ページの『Mode CLI/ODBC 構成キーワード』
- 443 ページの『SQLOverrideFileName CLI/ODBC 構成キーワード』
- 472 ページの『TxnIsolation CLI/ODBC 構成キーワード』
- 474 ページの『UseOldStpCall CLI/ODBC 構成キーワード』

db2cli.ini 初期設定ファイル

CLI/ODBC 初期設定ファイル (`db2cli.ini`) には、 CLI とこの製品を使うアプリケーションの動作を構成する場合に使用できる、さまざまなキーワードと値が入っています。

キーワードは、データベース別名と関連しており、そのデータベースにアクセスするすべての CLI および ODBC アプリケーションに影響を与えます。

作業を開始する助けとして `db2cli.ini.sample` サンプル構成ファイルが組み込まれています。`db2cli.ini.sample` ファイルに基づいて `db2cli.ini` ファイルを作成し、同じ場所にそれを保管することができます。サンプル構成ファイルの場所は、ご使用のドライバーのタイプおよびプラットフォームによって異なります。

IBM Data Server Client、IBM Data Server Runtime Client、または IBM Data Server Driver Package の場合、サンプル構成ファイルは次のいずれかのパスに作成されます。

- AIX®、HP-UX、Linux、または Solaris オペレーティング・システムの場合: *installation_path*`/cfg`
- Windows XP および Windows Server 2003 の場合: `C:¥Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥`*driver_copy_name*`¥cfg`

- Windows Vista および Windows Server 2008 の場合:
  C:¥ProgramData¥IBM¥DB2¥*driver_copy_name*¥cfg

例えば、 IBM Data Server Driver Package for Windows XP を使用していて、データ・サーバー・ドライバーのコピー名が IBMDBCL1 である場合、db2cli.ini.sample ファイルは C:¥Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥IBMDBCL1¥cfg ディレクトリーに作成されます。

IBM Data Server Driver for ODBC and CLI の場合、サンプル構成ファイルは次のいずれかのパスに作成されます。

- AIX、HP-UX、Linux、または Solaris オペレーティング・システムの場合:
  installation_path/cfg
- Windows の場合: *installation_path*¥cfg

  ここで、*installation_path* はドライバー・ファイルが抽出されたファイル・パスです。

例えば、IBM Data Server Driver for ODBC and CLI for Windows Vista を使用していて、ドライバーが C:¥IBMDB2¥CLIDRIVER¥V97FP3 ディレクトリーにインストールされている場合、db2cli.ini.sample ファイルは C:¥IBMDB2¥CLIDRIVER¥V97FP3¥cfg ディレクトリーに作成されます。

Windows オペレーティング・システム上でユーザー DSN を構成するために ODBC Driver Manager が使用されている場合、db2cli.ini ファイルは Documents and Settings¥User Name に作成されます。ここで、*User Name* はユーザー・ディレクトリーの名前を表します。

環境変数 **DB2CLIINIPATH** を使用して、db2cli.ini ファイルの別の場所を指定することができます。

構成キーワードを使用すると、以下のことが可能になります。

- データ・ソース名、ユーザー名、およびパスワードなどの一般的なフィーチャーを構成する。
- パフォーマンスに影響を及ぼすオプションを設定する。
- ワイルドカード文字などの照会パラメーターを指示する。
- さまざまな ODBC アプリケーション用にパッチまたは作業環境を設定する。
- コード・ページと IBM GRAPHIC データ・タイプなどの接続に関連したその他のフィーチャーを設定する。
- アプリケーションによって指定されるデフォルト接続オプションをオーバーライドする。例えば、アプリケーションが SQL_ATTR_ANSI_APP 接続属性を設定することによって CLI ドライバーに対して Unicode サポートを要求している場合でも、db2cli.ini ファイルの中で **DisableUnicode**=1 が設定されていると、CLI ドライバーはそのアプリケーションに Unicode サポートを提供しません。

  注: db2cli.ini ファイルの中で設定されている CLI/ODBC 構成キーワードが、SQLDriverConnect() 接続ストリングに含まれるキーワードと矛盾する場合、SQLDriverConnect() キーワードが優先されます。

db2cli.ini 初期設定ファイルは、 CLI 構成オプション用の値を保管している ASCII ファイルです。作業を開始する助けとして、サンプル・ファイルが組み込ま

れています。ほとんどの CLI/ODBC 構成キーワードは db2cli.ini 初期設定ファイル内に設定されますが、一部のキーワードはその代わりに、SQLDriverConnect() への接続ストリング内にキーワード情報を指定することによって設定されます。

ファイル内には、ユーザーが構成を希望するデータベース (データ・ソース) ごとに 1 つのセクションがあります。必要であれば、すべてのデータベース接続に影響を与える共通セクションもあります。

COMMON セクションには、 CLI/ODBC ドライバーを介したすべてのデータベース接続に適用するキーワードのみ含まれています。それには以下のキーワードが含まれます。

- **CheckForFork**
- **DiagPath**
- **DisableMultiThread**
- **JDBCTrace**
- **JDBCTraceFlush**
- **JDBCTracePathName**
- **QueryTimeoutInterval**
- **ReadCommonSectionOnNullConnect**
- **Trace**
- **TraceComm**
- **TraceErrImmediate**
- **TraceFileName**
- **TraceFlush**
- **TraceFlushOnError**
- **TraceLocks**
- **TracePathName**
- **TracePIDList**
- **TracePIDTID**
- **TraceRefreshInterval**
- **TraceStmtOnly**
- **TraceTime**
- **TraceTimeStamp**

他のすべてのキーワードはデータベース固有のセクションに置かれるようになっています。

注: 構成キーワードは COMMON セクション中で有効になりますが、すべてのデータベース接続に適用されます。

db2cli.ini ファイルの COMMON セクションは、次の語で始まります。

```
[COMMON]
```

共通キーワードを設定する前に、クライアントからのすべての CLI/ODBC 接続にこの設定が与える影響を評価するのは重要なことです。例えば、**TRACE** などのキーワ

ードは、DB2 に接続している CLI/ODBC アプリケーションのうち 1 つだけをトラブルシューティングしようとしている場合でも、これらのすべてのアプリケーションに関する情報をそのクライアントで生成します。

それぞれのデータベースの特定のセクションは、必ず大括弧で囲まれたデータ・ソース名 (DSN) の名前で始まります。

[*data source name***]**

これをセクション・ヘッダー と呼びます。

パラメーターを設定するには、キーワードとその関連キーワード値を次の形式で指定します。

KeywordName =*keywordValue*

- 各データベースのすべてのキーワードとその関連値は、そのデータベースのセクション・ヘッダーの下になければなりません。
- データベース固有のセクションに **DBAlias** キーワードが含まれていない場合は、接続が確立される際にはデータ・ソース名がデータベース別名として使用されます。各セクションのキーワード設定値は、該当するデータベース別名だけに適用されます。
- キーワードは大文字小文字の区別はありません。しかし、その値が文字ベースのものであれば値にその区別がある場合もあります。
- .INI ファイルにデータベースがない場合、これらのキーワードのデフォルト値が有効になっています。
- 新しい行の先頭位置にセミコロンを入れると、注釈行になります。
- ブランク行は許可されています。
- 1 つのキーワードに重複項目があると、最初の項目が使用されます (警告は与えられません)。

2 つのデータベース別名セクションがある .INI サンプル・ファイルを次に示します。

```
; This is a comment line.
[MYDB22]
AutoCommit=0
TableType="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CurrentSQLID=SAAID
TableType="'TABLE'"
SchemaList="'USER1',CURRENT SQLID,'USER2'"
```

db2cli.ini ファイルはすべてのプラットフォームで手動で編集できますが、使用できるなら **UPDATE CLI CONFIGURATION** コマンドを使用することをお勧めします。手作業で db2cli.ini ファイルを編集する場合、最後の項目の後にブランク行を追加してください。

AllowGetDataLOBReaccess CLI/ODBC 構成キーワード

アプリケーションが Dynamic Data Format をサポートするデータベース・サーバーを照会するときに、以前にアクセスした LOB 列に対して SQLGetData() を呼び出せるかどうかを指定します。

db2cli.ini キーワード構文:

AllowGetDataLOBReaccess = **0** | **1**

デフォルト設定:

アプリケーションが Dynamic Data Format をサポートするデータベース・サーバーを照会するときに、以前にアクセスした LOB 列に対して SQLGetData() を呼び出すことを許可しません。

使用上の注意:

このキーワードは、Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするデータベース・サーバーへの接続にのみ影響を与えます。デフォルト設定の 0 は、アプリケーションが以前にアクセスした LOB 列に対して SQLGetData() を呼び出すことを許可しません。アプリケーションが以前にアクセスした LOB 列に対して SQLGetData() を呼び出すことを許可するには、 1 を指定してください。

LOB 列への再アクセスを許可するためにキーワードが 1 に設定された場合、サーバー上のいくつかのリソースは SQLGetData() が完了しても解放されないことがあることに注意してください。

サーバーが Dynamic Data Format をサポートしない場合、このキーワードには効果がなく、以前にアクセスされた LOB 列に対して SQLGetData() を呼び出すことは許可されます。

AllowInterleavedGetData という類似のキーワードがあります。これは、アプリケーションが、Dynamic Data Format をサポートするデータ・サーバーを照会するときに、以前にアクセスした LOB 列に対して SQLGetData() を呼び出し、 SQLGetData() への以前の呼び出し時のデータ・オフセット位置を維持できるようにします。特定の接続またはステートメントに対して AllowGetDataLOBReaccess および AllowInterleavedGetData の両方が設定されている場合、 AllowInterleavedGetData 設定のほうが、AllowGetDataLOBReaccess よりも優先されます。

AllowInterleavedGetData CLI/ODBC 構成キーワード

アプリケーションが、Dynamic Data Format をサポートするデータ・サーバーを照会するときに、以前にアクセスした LOB 列に対して SQLGetData() を呼び出し、SQLGetData() への以前の呼び出し時のデータ・オフセット位置を維持できるかどうかを指定します。

db2cli.ini キーワード構文:

AllowInterleavedGetData = **0** | **1**

デフォルト設定:

アプリケーションが Dynamic Data Format をサポートするデータベース・サーバーを照会するときに、以前にアクセスした LOB 列に対して SQLGetData() を呼び出すことを許可しません。

同等の環境または接続属性:

SQL_ATTR_ALLOW_INTERLEAVED_GETDATA

使用上の注意:

このキーワードは、Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするデータベース・サーバーへの接続にのみ影響を与えます。デフォルト設定の 0 は、アプリケーションが以前にアクセスした LOB 列に対して SQLGetData() を呼び出すことを許可しません。 1 を指定すると、アプリケーションは以前にアクセスした LOB 列に対して SQLGetData()を呼び出し、以前の読み取り時にアプリケーションが読み取りを停止したところから LOB データの読み取りを開始することができます。

LOB 列への再アクセスを許可するためにキーワードが 1 に設定された場合、サーバー上のいくつかのリソースは SQLGetData() が完了しても解放されないことがあることに注意してください。

サーバーが Dynamic Data Format をサポートしない場合、このキーワードには効果がなく、以前にアクセスされた LOB 列に対して SQLGetData() を呼び出すことは許可されます。

AllowGetDataLOBReaccess という類似のキーワードがあります。これは、アプリケーションが、以前にアクセスした LOB 列に対して SQLGetData() を呼び出せるようにします。しかし、AllowGetDataLOBReaccess キーワードが使用されている場合、データ位置およびオフセット情報は維持されません。中断後に LOB 列が再度アクセスされるとき、SQLGetData() はその LOB データ列の先頭からデータの読み取りを開始します。特定の接続またはステートメントに対して AllowGetDataLOBReaccess および AllowInterleavedGetData の両方が設定されている場合、AllowInterleavedGetData 設定のほうが、AllowGetDataLOBReaccess よりも優先されます。

AltHostName CLI/ODBC 構成キーワード

HOSTNAME で指定された 1 次サーバーと通信できない場合に使用される代替ホスト名を指定します (クライアント・リルート)。

db2cli.ini キーワード構文:

AltHostName = *完全修飾された代替ホスト名 | ノードの IP アドレス*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、データベースの代替サーバーの常駐場所を示す、ノードの完全修飾ホスト名または IP アドレスを指定します。

1 次サーバーが代替サーバー情報を戻す場合、その情報はこの AltHostName 設定値をオーバーライドします。なお、このキーワードは読み取り専用です。つまり、1 次サーバーから受け取った代替サーバー情報によって db2cli.ini が更新されることはありません。

AltPort CLI/ODBC 構成キーワード

HOSTNAME および PORT で指定された 1 次サーバーと通信できない場合に使用される代替ポートを指定します (クライアント・リルート)。

db2cli.ini キーワード構文:

AltPort = *ポート番号*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、データベースの代替サーバーの常駐場所を示す、データベース・マネージャー・インスタンスの代替サーバーのポート番号を指定します。

1 次サーバーが代替サーバー情報を戻す場合、その情報はこの AltPort 設定値をオーバーライドします。なお、このキーワードは読み取り専用です。つまり、1 次サーバーから受け取った代替サーバー情報によって db2cli.ini が更新されることはありません。

AppUsesLOBLocator CLI/ODBC 構成キーワード

アプリケーションが LOB ロケーターを使用するかどうかを指定します。

db2cli.ini キーワード構文:

AppUsesLOBLocator = **0** | **1**

デフォルト設定:

アプリケーションは LOB ロケーターを使用しています。

同等の接続またはステートメント属性:

SQL_ATTR_APP_USES_LOB_LOCATOR

使用上の注意:

デフォルト設定の 1 は、アプリケーションが LOB ロケーターを使用していることを示します。 LOB ロケーターを使用しないアプリケーションで Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするサーバー上のデータを照会している場合は、0 を指定することによって、LOB ロケーターが使用されないことを示し、LOB データの戻りが最適化されるようにしてください。

ストアード・プロシージャーの結果セットの場合、このキーワードは無視されます。

キーワードが 0 に設定されている場合、アプリケーションが SQLBindCol() を使用して LOB ロケーターを結果セットにバインドすると、SQLFetch() 関数から無効な変換エラーが戻されます。

AppendAPIName CLI/ODBC 構成キーワード

エラーを生成した CLI/ODBC 関数名をエラー・メッセージ・テキストに付加します。

db2cli.ini キーワード構文:

AppendAPIName = **0** | **1**

デフォルト設定:

CLI 関数名を表示しません。

使用上の注意:

エラーを生成した CLI 関数 (API) 名は、 SQLGetDiagRec() または SQLError() を使用して検索されたエラー・メッセージに付加されます。関数名は中括弧 { } で囲まれます。

例えば、次のようになります。

```
[IBM][CLI Driver]" CLIxxxx: < text >
SQLSTATE=XXXXX {SQLGetData}"
```

- 0 = CLI 関数名を付加しません (デフォルト)
- 1 = CLI 関数名を付加します

このキーワードはデバッグにのみ役立ちます。

AppendForFetchOnly CLI/ODBC 構成キーワード

節 FOR FETCH ONLY を READ-ONLY SQL ステートメントに追加するかどうかを指定します。

db2cli.ini キーワード構文:

AppendForFetchOnly = **0** | **1**

デフォルト設定:

このキーワードは、デフォルトでは設定されません。 CLI は、特定のサーバー・タイプに接続されるときにのみ "FOR FETCH ONLY" 節を追加します。

同等の接続属性:

SQL_ATTR_APPEND_FOR_FETCH_ONLY

使用上の注意:

デフォルトで、CLI は、DB2 for z/OS または DB2 for i データベースに接続する際に、読み取り SELECT ステートメントに、"FOR FETCH ONLY" 節を追加します。

このキーワードにより、アプリケーションは CLI が FOR FETCH ONLY 節をいつ追加するかを制御できます。例えば、アプリケーションがさまざまなバインドの BLOCKING オプション (BLOCKING UNAMBIG など) を使用して CLI パッケージをバインドしていて、特定の行に存続するためにブロッキングを抑制する必要がある場合などに、このキーワードを使用できます。

デフォルトの CLI 動作を変更するには、キーワードを次のように設定できます。

- 0: CLI は、接続先のサーバー・タイプには関係なく、読み取り SELECT ステートメントに FOR FETCH ONLY 節を追加しません。

- 1: CLI は、接続先のサーバー・タイプには関係なく、読み取り SELECT ステートメントに FOR FETCH ONLY 節を追加します。

AppendRowColToErrorMessage CLI/ODBC 構成キーワード

エラーを生成した行および列の番号を、エラー・メッセージ・ストリングに追加するかどうかを指定します。

db2cli.ini キーワード構文:

AppendRowColToErrorMessage= **0** | **1**

デフォルト設定:

デフォルトの設定値 0 は、行および列の番号を付けずに、エラー・メッセージ・ストリングを戻します。

使用上の注意:

1 を指定すると、エラーを生成した行および列の番号を、エラー・メッセージ・ストリングに追加します。行および列の番号の値が追加されるのは、DB2 CLI が行または列の番号を問題に適合できる場合だけです。

エラー・メッセージに追加される行または列の番号は、アプリケーションが SQLGetDiagField() を DiagIdentifier 引数 SQL_DIAG_ROW_NUMBER または SQL_DIAG_COLUMN_NUMBER を指定して呼び出した場合に戻されるのと同じ正の値です。 AppendRowColToErrorMessage が 1 に設定されると、SQLGetDescField()、SQLGetDescRec()、または SQLError() の呼び出しから戻されるエラーには、これらの行番号または列番号 (判別できる場合) が、次の形式で追加されます。Row=<r>, Col=<c>

例えば、エラー CLI0111E のデフォルトのテキストは、次のとおりです。

```
[IBM][CLI Driver] CLI0111E  Numeric value out of range. SQLSTATE=22003
```

行および列の番号を追加する 1 を指定すると、エラー CLI0111E では以下のテキストが戻されます。

```
[IBM][CLI Driver] CLI0111E  Numeric value out of range.
   SQLSTATE=22003 {Row=2,Col=1}
```

注: 行番号だけが追加されて、エラーが戻されることもあります。

ArrayInputChain CLI/ODBC 構成キーワード

配列入力を有効にします。通常の配列入力の場合とは異なり、事前にサイズやメモリー割り振り要件を指定する必要はありません。

db2cli.ini キーワード構文:

ArrayInputChain = **-1** | **0** | **<positive integer>**

デフォルト設定:

通常の入力配列が有効になります。この場合、対応する SQLExecute() 呼び出しが実行される前に、配列とそのサイズを指定する必要があります。

使用上の注意:

デフォルトでは、配列入力 (値の配列が入力パラメーターにバインドされている) の場合、対応する SQLExecute() 関数が呼び出される前に、配列とそのサイズが指定

されていなければなりません。しかし、アプリケーションにおいて配列のサイズが事前にはわからない場合や、配列のサイズが大きいため、使用可能メモリーのプールからアプリケーションが割り振ることができない場合があります。そのような場合、アプリケーションでは ArrayInputChain=-1 を設定し、SQL_ATTR_CHAINING_BEGIN および SQL_ATTR_CHAINING_END のステートメント属性を使用することによって、チェーニングを有効にすることができます。その場合、通常の配列入力の場合とは異なり、事前にサイズやメモリー要件を指定することなく配列入力を利用できます。

チェーニングを有効にするには、

1. keyword ArrayInputChain = -1 を設定します。
2. 入力パラメーターを準備し、SQL ステートメントにバインドします。
3. SQLSetStmtAttr() により SQL_ATTR_CHAINING_BEGIN ステートメント属性を設定します。
4. バインドされたパラメーターを入力データにより更新し、 SQLExecute() を呼び出します。
5. 入力配列に含まれる行の数だけステップ 4 を繰り返します。
6. ステップ 4 で配列の最後の行が処理された後、 SQLSetStmtAttr() を使用して SQL_ATTR_CHAINING_END ステートメント属性を設定します。

これらのステップを完了した結果は、通常の配列入力を使用した場合と同じになります。

ArrayInputChain=0 (デフォルト値) を設定すると、この配列入力フィーチャーがオフになります。 ArrayInputChain には、入力配列に対して使用する配列サイズを設定するための任意の正整数を設定することもできます。

制約事項: DB2 CLI はコンパウンド SQL (コンパイル済み) ステートメントまたはコンパウンド SQL (インライン) ステートメントでの配列入力チェーニングをサポートしていません。

AsyncEnable CLI/ODBC 構成キーワード

照会を非同期に実行する機能を、有効または無効に設定します。

db2cli.ini キーワード構文:

AsyncEnable = **0** | **1**

デフォルト設定:

照会を非同期に実行できます。

使用上の注意:

このオプションによって、照会を非同期に実行できるサポートを、有効または無効に設定できます。これは、SQLSetStmtAttr() または SQLSetConnectAttr() を使用して SQL_ATTR_ASYNC_ENABLE 属性を設定することによって、このフィーチャーを利用するように作成されたアプリケーションにのみ有効となります。

- 0 = 照会が非同期に実行されません。
- 1 = 照会が非同期に実行されるのを許可します。アプリケーションはまた、SQLSetStmtAttr() または SQLSetConnectAttr() を使用して

SQL_ATTR_ASYNC_ENABLE を設定することによって、非同期機能を有効にする必要があります。 (デフォルト)

関数が非同期に呼び出されると、元の関数が SQL_STILL_EXECUTING 以外のコードを戻すまでは、元の関数、 SQLAllocHandle()、SQLCancel()、SQLSetStmtAttr()、SQLGetDiagField()、SQLGetDiagRec()、または SQLGetFunctions() だけが、ステートメント・ハンドルで呼び出せます。同じ接続下にある他のステートメント・ハンドルで他の関数が呼び出されると、SQL_ERROR が SQLSTATE HY010 (関数シーケンス・エラー) を伴って戻されます。

Attach CLI/ODBC 構成キーワード

サーバー・インスタンスにアタッチするかどうかを指定します。このキーワードを接続ストリング内で指定するか、db2cli.ini または db2dsdriver.cfg ファイルで設定することができます。

db2cli.ini キーワード構文:

ATTACH = **TRUE** | **FALSE**

デフォルト設定:

SQLDriverConnect() 関数は、指定のデータベースに接続します。

同等の環境または接続属性:

N/A

使用上の注意:

このキーワードを TRUE に設定すると、SQLDriverConnect() 関数はデータベースには接続しませんが、代わりに指定のサーバー・インスタンスに接続します。

Linux、UNIX、および Windows リモート・サーバー用の DB2 サーバー・インスタンスへの接続を確立するには、CLI アプリケーションで、このキーワードを TRUE に設定するとともに、Hostname、Port、UID、PWD、Protocol の値も指定する必要があります。

TRUE 以外の値を **ATTACH** キーワードに割り当てても、すべて FALSE として扱われます。

例: 以下の例は、db2cli.ini ファイルでのキーワードの指定を示しています。

```
ATTACH=TRUE
```

以下の例は、db2dsdriver.cfg ファイルでのキーワードの指定を示しています。

```
<configuration>
   <dsncollection>
        <dsn
alias="db2dsn01",name="db2db01",host="server1.mynet.com",
port="50001">
           <parameter name="ATTACH" value="TRUE"/>
        </dsn>
    </dsncollection>
    <databases>
        <database name="sample", host="serv1.mynet.com",
port="50001">
```

```
        <parameter name="ATTACH" value="TRUE"/>
      </database>
    </databases>
</configuration>
```

バージョン情報

最終更新

このトピックの最終更新対象は、IBM DB2 バージョン 9.7 フィックスパック 3 です。

IBM Data Server Client

IBM DB2 Database for Linux, UNIX, and Windows でサポート

Authentication CLI/ODBC 構成キーワード

ファイル DSN 接続または DSN なし接続で使用される認証タイプを指定します。

db2cli.ini キーワード構文:

Authentication = CERTIFICATE | SERVER | SERVER_ENCRYPT | SERVER_ENCRYPT_AES | DATA_ENCRYPT | KERBEROS | GSSPLUGIN

デフォルト設定:

SERVER

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このオプションを設定するとき、以下のオプションもまた設定する必要があります。

- **Database**
- **Protocol**

Protocol=IPC の場合には、以下のオプションもまた設定する必要があります。

- **Instance**

Protocol=TCPIP の場合には、以下のオプションもまた設定する必要があります。

- **Port**
- **Hostname**

Kerberos を指定する場合には、オプションで **KRBPlugin** を指定することもできます。**KRBPlugin** を指定しない場合、デフォルト・プラグイン IBMkrb5 が使用されます。

DB2 バージョン 9.7 フィックスパック 6 以降、APAR PM53450 以降が適用された DB2 for z/OS バージョン 10 との接続に証明書認証を使用できるようになっています。CERTIFICATE 認証タイプは、DB2 バージョン 9.7 フィックスパック 6 からサポートされています。この認証タイプでは、SSL クライアント認証を使用でき、データベース・クライアントでデータベース・パスワードを提供する必要がありません。証明書ベースの認証を構成して認証情報を指定すると、他の方法 (db2dsdriver.cfg 構成ファイル、

db2cli.ini 構成ファイル、または接続ストリングに指定するという方法) ではパスワードを指定できなくなります。 CERTIFICATE を指定する場合は、CLI 構成ファイル db2cli.ini またはデータ・サーバー・ドライバー構成ファイル db2dsdriver.cfg に、新しいラベル・パラメーター SSLCLientLabel を指定する必要もあります。

AutoCommit CLI/ODBC 構成キーワード

デフォルトでアプリケーションがステートメントごとにコミットするかどうかを指定します。

db2cli.ini キーワード構文:

AutoCommit = **1** | **0**

デフォルト設定:

各ステートメントは、単一の完全なトランザクションとして扱われます。

同等の接続属性:

SQL_ATTR_AUTOCOMMIT

使用上の注意:

ODBC との整合性を保つために、CLI では AutoCommit のデフォルトはオンになっています。つまり、各ステートメントは単一の完全なトランザクションとして扱われます。このキーワードでは別のデフォルトを提供することができますが、アプリケーションが、SQL_ATTR_AUTOCOMMIT の値を指定しない場合にのみ使用してください。

- 1 = SQL_ATTR_AUTOCOMMIT_ON (デフォルト)
- 0 = SQL_ATTR_AUTOCOMMIT_OFF

注: ほとんどの ODBC アプリケーションでは、 AutoCommit のデフォルトはオンであるということを前提にしています。アプリケーションはこのデフォルトの下で正しく機能する場合があるので、実行時の間にこのデフォルトをオーバーライドする場合は、十分に注意する必要があります。

このキーワードにより、分散作業単位 (DUOW) 環境で自動コミットを有効にするかどうかを指定することもできます。接続が整合分散作業単位の一部であり、AutoCommit が設定されないと、デフォルトは適用されません。自動コミット処理から生じる暗黙的コミットは抑制されます。 AutoCommit が 1 に設定され、接続が整合分散作業単位の一部である場合は、暗黙的コミットが処理されます。これが重大な性能低下につながり、DUOW システム外で他の予期しない結果が生じる可能性があります。ただし、これが有効でなければ、一部のアプリケーションはまったく機能しない可能性があります。

アプリケーションのトランザクション処理に関して、特にサード・パーティーによって作成されたアプリケーションについては、これを DUOW 環境に適用する前に、完全に理解する必要があります。

BIDI CLI/ODBC 構成キーワード

DB2 for z/OS に接続される場合に BIDI コード・ページを指定します。

db2cli.ini キーワード構文:

BIDI = *code page*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このオプションを設定する場合は、以下のオプションも設定する必要があります。

- Database
- Protocol=TCPIP
- Hostname
- Port

BitData CLI/ODBC 構成キーワード

バイナリー・データ・タイプをバイナリー・データ・タイプとして報告するか、文字データ・タイプとして報告するかを指定します。

db2cli.ini キーワード構文:

BitData = **1** | **0**

デフォルト設定:

FOR BIT DATA および BLOB データ・タイプをバイナリー・データ・タイプとして報告します。

使用上の注意:

このオプションを使用すると、 ODBC バイナリー・データ・タイプ (SQL_BINARY、SQL_VARBINARY、 SQL_LONGVARBINARY、および SQL_BLOB) をバイナリー・タイプ・データとして報告するかどうかを指定できます。 FOR BIT DATA 属性を持つ CHAR、VARCHAR、および LONG VARCHAR 列を定義することによって、 IBM DBMS はバイナリー・データ・タイプの列をサポートします。 DB2 Database for Linux, UNIX, and Windows は BLOB を介してバイナリー・データもサポートします (その場合、このデータは CLOB データ・タイプにマップされます)。

FOR BIT DATA または BLOB として定義されたすべての列に文字データのみが含まれ、アプリケーションがバイナリー・データ列を表示できないことが確実な場合にのみ、 BitData = 0 を設定してください。

- 1 = FOR BIT DATA および BLOB データ・タイプをバイナリー・データ・タイプとして報告します (デフォルト)。
- 0 = FOR BIT DATA および BLOB データ・タイプを文字データ・タイプとして報告します。

BlockForNRows CLI/ODBC 構成キーワード

1 回のフェッチで戻されるデータ行の数を指定します。

db2cli.ini キーワード構文:

BlockForNRows = <正の整数>

デフォルト設定:

サーバーは、1 回のフェッチ要求に対して、 1 つの照会ブロックに入るだけの数の行を戻します。

使用上の注意:

BlockForNRows キーワードは、 1 回のフェッチ要求でクライアントに戻されるデータ行の数を制御します。 BlockForNRows が指定されていない場合 (デフォルトの設定の場合)、 1 個の照会ブロックに入るだけの数の非 LOB データ行がサーバーから戻されます。結果セットに LOB データが含まれている場合の BlockForNRows の動作は、 BlockLobs CLI/ODBC 構成キーワード、および LOB データ・タイプを戻す結果セットのブロッキングがサーバーでサポートされているかどうかによって影響を受ける場合があります。

次のような場合に、1 つの照会ブロックに完全に入るだけの数の行に関連するすべての LOB データが、 1 回のフェッチ要求で戻されます。

- BlockForNRows が指定されていない。
- BlockLobs が 1 に設定されている。および
- サーバーが、LOB データ・タイプを戻す結果セットのブロッキングをサポートしている。

ここで LOB データが行に関連するものとして記述されているのは、結果セットの LOB データ自体は行に含まれていないからです。行に含まれるのは、実際の LOB データへの参照です。

BlockForNRows が正の整数 n に設定されている場合には、 1 回のフェッチ要求に対して n 個のデータ行が戻されます。結果セットに LOB データが含まれていて、LOB データ・タイプを戻す結果セットのブロッキングがサーバーでサポートされている場合、 n 個のデータ行に対応する LOB データが 1 回のフェッチ要求に対して戻されます。結果セットに LOB データが含まれているが、サーバーで LOB データ・タイプを戻す結果セットのブロッキングがサポートされていない場合には、1 回のフェッチ要求に対して戻されるのは LOB データを含む 1 行だけになります。

BlockLobs CLI/ODBC 構成キーワード

LOB ブロッキングをサポートするサーバーに対して、 LOB ブロッキング・フェッチを有効にします。

db2cli.ini キーワード構文:

BlockLobs = 0 | 1

デフォルト設定:

LOB データ・タイプを戻す結果セットのブロッキングは無効になっています。

同等のステートメント属性:

SQL_ATTR_BLOCK_LOBS

使用上の注意:

BlockLobs が 1 に設定されている場合、LOB ブロッキングがサーバーでサポートされているなら、1 つの照会ブロックに完全に入るだけの数の行に関連するすべての LOB データが、1 回のフェッチ要求で戻されます。**BlockLobs** = 1 を有効にして LOB 値をバッファーに直接バインドする CLI クライアントは、1 回の要求で取り出されるデータの量を以前のリリースと比較した結果によって、メモリー消費量の増加を表示することができます。ここで LOB データが行に関連するものとして記述されているのは、結果セットの LOB データ自体は行に含まれていないからです。行に含まれるのは、実際の LOB データへの参照です。したがって、LOB データ・タイプを戻す結果セットのブロッキングを使用する場合、サーバーで LOB データ・タイプを戻す結果セットのブロッキングがサポートされているなら、結果セットのうち照会ブロック内に完全に入るだけの数の行が、サーバーから戻される LOB データに関連付けられることになります (LOB データは行に直接格納されるわけではないため、各行の内容は非 LOB データです)。

サーバーで LOB 列を使用したカーソル・ブロッキングがサポートされていない場合、1 回のフェッチ要求で戻されるのは 1 行の LOB データのみで、**BlockLobs** 値は無視されます。DB2 Database for Linux, UNIX, and Windows は LOB 列を使用したカーソル・ブロッキングをサポートしていますが、他のサーバーではサポートしていない可能性があります。

DB2 Database for Linux, UNIX, and Windows は LOB ブロッキング・フェッチをサポートしません。

IDS データ・サーバーは、 LOB ブロッキング・フェッチをサポートしていません。

CLIPkg CLI/ODBC 構成キーワード

生成するラージ・パッケージの数を指定します。

db2cli.ini キーワード構文:

CLIPkg = **3** | **4** | ... | **30**

デフォルト設定:

3 つの大きなパッケージが生成されます。

使用上の注意:

このキーワードは、CLI/ODBC アプリケーションでの SQL セクションの数を増やすのに使用されます。これが使用される場合には、管理者は CLIPkg BIND オプションを使用して、必要なバインド・ファイルを明示的にバインドする必要があります。クライアント・アプリケーションの場合、クライアント上の db2cli.ini ファイルは、この CLIPkg の値で更新する必要があります。 CLI/JDBC ストアード・プ

ロシージャーの場合、 (UNIX または Intel プラットフォームでの DB2 UDB バージョン 6.1 以降の) サーバー上の db2cli.ini ファイルは、CLIPkg の同じ値で更新する必要があります。

値として 3 以上 30 以下の整数が指定されていない場合には、デフォルトが使用されます。このとき、エラーまたは警告は出されません。

この設定は、ラージ・パッケージ (384 個のセクションを含む) にのみ適用されます。スモール・パッケージ (64 個のセクションを含む) の数は 3 個であり、変更できません。

パッケージはデータベースでスペースをとるため、増やすセクションの数は、ご使用のアプリケーションを実行できるだけの数にとどめるようお勧めします。

CheckForFork CLI/ODBC 構成キーワード

関数呼び出しごとに、fork されたプロセスを検査します。

db2cli.ini キーワード構文:

0 | **1**

デフォルト設定:

CLI は、fork されたプロセスを検査しません。

使用上の注意:

CLI は、プロセスが決して fork されないと想定します。接続およびステートメント・ハンドルが割り振られる際、親プロセスのアクティブな接続と干渉することを回避するためにアプリケーションが fork することを望む場合、**CheckForFork** キーワードを 1 に設定する必要があります。

SQL_ATTR_PROCESSCTL 環境属性は、アプリケーションに対する **CheckForFork** キーワードをオーバーライドするために、そのアプリケーションによって SQL_PROCESSCTL_NOTHREAD オプションに設定されることがあります。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 データベースへのすべての接続に適用されます。)

ClientAcctStr CLI/ODBC 構成キーワード

データベースに送信されるクライアント・アカウンティング・ストリングを設定します。

db2cli.ini キーワード構文:

ClientAcctStr = *アカウンティング・ストリング*

デフォルト設定:

なし

次の場合に適用可能:

DB2 Connect または DB2 Database for Linux, UNIX, and Windows を使用してデータベースに接続しているとき

同等の環境または接続属性:

SQL_ATTR_INFO_ACCTSTR

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect または DB2 データベース製品を介してデータベースに送られたクライアント・アカウンティング・ストリングを設定できます。デフォルトでアカウンティング・ストリングを提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 200 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

ClientApplName CLI/ODBC 構成キーワード

データベースに送信されるクライアント・アプリケーション名を設定します。

db2cli.ini キーワード構文:

ClientApplName = *アプリケーション名*

デフォルト設定:

なし

次の場合に適用可能:

DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品を使用してデータベースに接続しているとき

同等の環境または接続属性:

SQL_ATTR_INFO_APPLNAME

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect または DB2 データベース製品を介してデータベースに送られたクライアント・アプリケーション名を設定できます。デフォルトでアプリケーション名を提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 32 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

ClientBuffersUnboundLOBS CLI/ODBC 構成キーワード

アプリケーション・パラメーターにバインドされていない LOB 列に対して、 LOB ロケーターの代わりに LOB データをフェッチするかどうかを指定します。

db2cli.ini キーワード構文:

ClientBuffersUnboundLOBS = **0** | **1**

デフォルト設定:

アプリケーション・パラメーターにバインドされていない LOB 列に対して、実際の LOB データの代わりに LOB ロケーターが検索されます。

使用上の注意:

デフォルトでは、アプリケーション・パラメーターにバインドされていない LOB 列が結果セットに含まれている場合、 CLI は LOB データそのものではなく対応する LOB ロケーターをフェッチします。その場合、アプリケーションは LOB データを取り出すのに `SQLGetLength()`、`SQLGetPosition()`、および `SQLGetSubString()` CLI 関数を使用しなければなりません。アプリケーションが定期的に LOB データの検索を要求する場合は、このデフォルトの 2 段階処理は不要であり、効率は低下する可能性があります。この場合、ClientBuffersUnboundLOBS = 1 を設定して、強制的に DB2 CLI が LOB ロケーターではなく LOB データをフェッチするようにします。

Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするサーバーは、LOB および XML データの戻りをデータの実際の長さに応じて最適化します。 LOB および XML データは、その全体を戻すか、またはプログレッシブ参照と呼ばれる内部トークンとして戻すことができます。 CLI はプログレッシブ参照のデータ検索を管理します。

Dynamic Data Format をサポートするサーバー上のデータを照会するアプリケーションでは、LOBCacheSize キーワードを設定すると、データの全体を戻すかまたはプログレッシブ参照として戻すかを決定するために使用されるしきい値が設定されます。データの長さが LOBCacheSize しきい値よりも大きい場合、プログレッシブ参照が管理のために CLI に戻されますが、データの長さが LOBCacheSize しきい値以下の場合は、データの全体が戻されます。 ClientBuffersUnboundLOBS を 1 に設定すると、LOBCacheSize を 2147483647 に設定したのと同じことになり、データはプログレッシブ参照としてではなくその全体がサーバーから戻されることになります。

ClientEncAlg CLI/ODBC 構成キーワード

ユーザー ID とパスワードを暗号化するときに使用する暗号化アルゴリズムのタイプを指定します。

db2cli.ini キーワード構文:

ClientEncAlg = 1 | 2 | AES

デフォルト設定:

任意の暗号化アルゴリズムを使用できます。

次の場合に適用可能:

リモート・データベースへの接続時。

同等の環境または接続属性:

SQL_ATTR_CLIENT_ENCALG

使用上の注意:

このキーワードの値は次のように定義されています。

- 1 - 任意の暗号化アルゴリズムを使用してユーザー ID とパスワードを暗号化します。
- 2 - Advanced Encryption Standard (AES) 暗号化アルゴリズムを使用してユーザー ID とパスワードを暗号化します。
- AES - 2 に相当します。

CLI 属性 SQL_ATTR_CLIENT_ENCALG はこのキーワードと動作が類似していますが、無効な属性値が指定された場合に「CLI0191E 属性の値が無効です」というエラーを返す点が異なります。CLI キーワードまたは接続属性値は、システム・データベース・ディレクトリーに指定された認証タイプに優先します。

ClientUserID CLI/ODBC 構成キーワード

データベースに送信されるクライアント・ユーザー ID を設定します。

db2cli.ini キーワード構文:

ClientUserID = *userid*

デフォルト設定:

なし

次の場合に適用可能:

DB2 Connect または DB2 Database for Linux, UNIX, and Windows を使用してデータベースに接続しているとき

同等の環境または接続属性:

SQL_ATTR_INFO_USERID

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect または DB2 データベース製品を介してデータベースに送られたクライアント・ユーザー ID (アカウンティング・ユーザー ID) を設定できます。デフォルトでユーザー ID を提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 16 文字までの長さです。
- このユーザー ID を認証ユーザー ID と混同しないでください。このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。

- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

ClientWrkStnName CLI/ODBC 構成キーワード

データベースに送信されるクライアント・ワークステーション名を設定します。

db2cli.ini キーワード構文:

ClientWrkStnName = ワークステーション名

デフォルト設定:

なし

次の場合に適用可能:

DB2 Connect または DB2 Database for Linux, UNIX, and Windows を使用してデータベースに接続しているとき

同等の環境または接続属性:

SQL_ATTR_INFO_WRKSTNNAME

使用上の注意:

このオプションによって、 CLI アプリケーションは DB2 Connect または DB2 データベース製品を介してデータベースに送られたクライアント・ワークステーション名を設定できます。デフォルトでクライアント・ワークステーション名を提供しないアプリケーションは、その情報を提供するためにこのキーワードを利用できます。

以下の条件に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 18 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

ColumnwiseMRI CLI/ODBC 構成キーワード

DB2 for z/OS サーバーに対して、配列入力チェーニングが列方向配列入力に変換されるかどうかを指定します。

db2cli.ini キーワード構文:

ColumnwiseMRI = **ON** | **OFF**

デフォルト設定:

配列入力チェーニングから列方向配列入力への変換は使用不可になっています。

同等の接続属性:

SQL_ATTR_COLUMNWISE_MRI

使用上の注意:

DB2 for z/OS の複数行挿入 (MRI) フィーチャーは、データが列方向配列の形式であることを想定します。したがって、データは圧縮された形式で送信されるため、この変換を使用して、配列入力チェーニング・フィーチャーを使用するアプリケーションのパフォーマンスを最適化することができます。以下の例では、この変換を使用可能にする方法を示します。

```
[dsn-name]
...
ColumnwiseMRI=ON
```

特定の場合に、変換は実行されません。このような場合について詳しくは、SQL_ATTR_COLUMNWISE_MRI を参照してください。

このキーワードは、DB2 for z/OS サーバーにのみ適用されます。

CommitOnEOF CLI/ODBC 構成キーワード

結果セットから最後の行を受け取った直後に、暗黙的な COMMIT を実行するかどうかを指定します。このキーワードを使用して、アプリケーションがカーソルから結果セット全体を受け取った直後に、リソースを解放できます。

db2cli.ini キーワード構文:

CommitOnEOF = **0** | **1**

デフォルト設定:

0

同等の接続属性:

SQL_ATTR_COMMITONEOF

使用上の注意:

この最適化の効果を生かすには、自動コミットを使用可能にし、カーソルを読み取り専用および前方スクロールにする必要があります。

ストアード・プロシージャーまたはアプリケーションが複数の結果セットを返した場合、最後のカーソルの結果セットから最後の行を読み取ったときに、COMMIT が実行されます。

ConcurrentAccessResolution CLI/ODBC 構成キーワード

使用する並行アクセス解決方法を指定します。

db2cli.ini キーワード構文:

ConcurrentAccessResolution = **0** | **1** | **2** | **3**

デフォルト設定:

DB2 CLI は準備オプションを提供しません。また、Currently Committed の動作はデータベース構成によって決定されます。

次の場合に適用可能:

DB2 Connect または DB2 Database for Linux, UNIX, and Windows を使用してデータベースに接続しているとき

同等の環境または接続属性:

SQL_ATTR_CONCURRENT_ACCESS_RESOLUTION

使用上の注意:

このキーワードは、カーソル固定 (CS) スキャンに対して指定されているデフォルト動作をオーバーライドする準備属性を指定します。

- 0 = 設定なし。クライアントは準備オプションを提供しません。
- 1 = Currently Committed セマンティクスを使用します。CLI は、準備が行われる度に「Currently Committed」をコマンド・フローの一部として送信します。つまり、データベース・マネージャーは、データが更新または削除の処理中であっても、該当するスキャンについてデータの現在コミット済みバージョンを使用することができます。挿入処理中の行はスキップすることができます。この設定は、実際の分離レベルがカーソル固定または読み取り固定のときに適用され (読み取り固定の場合はコミットされていない挿入のみをスキップ)、それ以外のときには無視されます。該当するスキャンには、読み取り専用スキャン (読み取り専用ステートメントの一部であることも、読み取り専用でないステートメントの一部であることもある) が含まれます。レジストリー変数 **DB2_EVALUNCOMMITTED**、**DB2_SKIPDELETED**、および **DB2_SKIPINSERTED** の設定は、currently committed を使用するスキャンには適用されません。ただし、これらのレジストリー変数の設定は、currently committed を使用しないスキャンには引き続き適用されます。
- 2 = 結果を待機します。CLI は、準備が行われる度に「結果の待機」をコマンド・フローの一部として送信します。つまり、カーソル固定以上のスキャンは、更新または削除の処理中のデータを検出した場合にコミットまたはロールバックを待機します。挿入処理中の行はスキップされません。レジストリー変数 DB2_EVALUNCOMMITTED、DB2_SKIPDELETED、および DB2_SKIPINSERTED の設定は適用されなくなりました。
- 3 = ロックされたデータをスキップします。CLI は、準備が行われる度に「ロックされたデータをスキップ」をコマンド・フローの一部として送信します。つまり、Currently Committed セマンティクスが使用され、挿入処理中の行はスキップされます。このオプションは、DB2 Database for Linux, UNIX, and Windows ではサポートされません。指定されていても、この設定は無視されます。

DB2 Database for Linux, UNIX, and Windows の場合、このキーワードを使用して、**cur_commit** 構成パラメーターによって定義された Currently Committed のデフォルトの動作をオーバーライドします。 DB2 for z/OS の場合、このキーワードを使用して、Currently Committed の動作を有効にします。この動作を指定することに相当する、DB2 for z/OS で使用可能なデータベース構成パラメーターはありません。

DB2 z/OS バージョン 10は、currently committed の INSERT 操作および DELETE 操作のみをサポートします。

ConnectNode CLI/ODBC 構成キーワード

接続が行われるデータベース・パーティション・サーバーを指定します。

db2cli.ini キーワード構文:

ConnectNode = **0 から 999 までの整数値 | SQL_CONN_CATALOG_NODE**

デフォルト設定:

マシンでポート 0 に定義されているデータベース・パーティション・サーバーが使用されます。

次の場合にのみ適用可能:

パーティション・データベース環境への接続時

同等の接続属性:

SQL_ATTR_CONNECT_NODE

使用上の注意:

これは、接続するターゲット・データベース・パーティション・サーバーを指定するために使用されます。以下のように設定できます。

- 0 から 999 までの整数
- SQL_CONN_CATALOG_NODE

この変数が設定されていない場合、ターゲットにはデフォルトとしてマシンのポート 0 に定義されたデータベース・パーティション・サーバーが使用されます。

このキーワード (または属性の設定) は、**DB2NODE** 環境変数の値をオーバーライドします。このキーワードの範囲外の指定値は無視され、代わりに SQL_CONN_CATALOG_NODE 値が使用されます。

ConnectTimeout CLI/ODBC 構成キーワード

試行を終了し通信タイムアウトを生成する前の、サーバーへの接続の確立を試行するときに応答を待機する時間を、秒単位で指定します。

db2cli.ini キーワード構文:

ConnectTimeout = 0 | 1 |2 | ... | 32767

デフォルト設定:

クライアントは接続の確立を試行するときに、サーバーからの応答を無制限に待機します。

同等の接続属性:

SQL_ATTR_LOGIN_TIMEOUT

使用上の注意:

ConnectTimeout が設定され、クライアント・リルートが有効な場合、接続は元のサーバーに対して一度だけ、さらに代替サーバーに対して一度試行されます。それぞれのサーバーへの接続試行で **ConnectTimeout** 値が使用されるので、最大待機時間は **ConnectTimeout** に指定された値の約 2 倍になります。どちらのサーバーも、キーワードで指定された時間内に到達できない場合には、以下のエラー・メッセージを受け取ります。

```
SQL30081N  A communication error has been detected.  Communication
   protocol being used: "TCP/IP". Communication API being used:
   "SOCKETS".  Location where the error was detected: "<ip address>".
   Communication function detecting the error: "<failing function>".
   Protocol specific error code(s): "<error code>", "*", "*".
   SQLSTATE=08001
```

ConnectTimeout が設定され、Sysplex 利用が有効な場合、接続は Sysplex メンバーのそれぞれに対して一度だけ試行されます。それぞれの Sysplex メ

ンバーへの接続試行で **ConnectTimeout** 値が使用されるので、最大待機時間は Sysplex メンバー数に、**ConnectTimeout** キーワードで指定された時間を掛けたものにほぼ等しくなります。

ConnectTimeout は TCPIP プロトコルだけに適用され、SOCKS 対応の TCP/IP ノード上でカタログされたデータベースへの接続に対してはサポートされません。

db2cli.ini ファイル内で明示的に指定された **ConnectTimeout** 値は SQL_ATTR_LOGIN_TIMEOUT よりも、実行時に優先されます。

ConnectType CLI/ODBC 構成キーワード

アプリケーションをリモート作業単位で実行するか、それとも分散作業単位で実行するかを制御します。

db2cli.ini キーワード構文:

ConnectType = **1** | **2**

デフォルト設定:

リモート作業単位。

同等の環境または接続属性:

SQL_ATTR_CONNECTTYPE

使用上の注意:

このオプションによって、デフォルトの接続タイプを指定できます。オプションは、次のとおりです。

- 1 = リモート作業単位。それぞれのコミット範囲がある、複数の同時接続。並行トランザクションは整合されていません。これはデフォルトです。
- 2 = 分散作業単位。複数のデータベースが同じ分散作業単位の下で参加する、整合接続。

最初の接続は、同じ環境ハンドルの下に割り振られた他のすべて接続の接続タイプを決定します。

このキーワードは、環境または接続属性よりも優先されます。

CurrentFunctionPath CLI/ODBC 構成キーワード

動的 SQL ステートメントで関数参照およびデータ・タイプ参照の解決に使用されるスキーマを指定します。

db2cli.ini キーワード構文:

CurrentFunctionPath = *current_function_path*

デフォルト設定:

以下の記述を参照してください。

使用上の注意:

このキーワードは、動的 SQL で使用される関数参照およびデータ・タイプ参照の解決に使用されるパスを定義します。これには、1 つ以上のスキーマ名のリストが含まれます。スキーマ名は二重引用符で囲まれ、コンマで区切られます。

デフォルト値は "SYSIBM","SYSFUN",X です。X は、二重引用符で区切られた、USER 特殊レジスターの値です。スキーマ SYSIBM を指定する必要はありません。関数パスに組み込まれていない場合、暗黙的にこのスキーマが最初のスキーマであると見なされます。

このキーワードは、現在のユーザーのスキーマ以外のスキーマ名で定義された可能性のある非修飾関数およびストアード・プロシージャー参照を解決する処理の一部として使用されます。スキーマ名の順序によって、関数名およびプロシージャー名が解決される順序が決定されます。

CurrentImplicitXMLParseOption CLI/ODBC 構成キーワード

CURRENT IMPLICIT XMLPARSE OPTION 特殊レジスターの値を設定します。

db2cli.ini キーワード構文:

CurrentImplicitXMLParseOption = **'STRIP WHITESPACE'** | **'PRESERVE WHITESPACE'**

デフォルト設定:

妥当性検査のない暗黙的な構文解析の際に、空白文字は除去されます。

同等の接続属性:

SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION

使用上の注意:

このキーワードを設定すると、データベースへの接続ごとに SET CURRENT IMPLICIT XMLPARSE OPTION ステートメントが発行されます。デフォルトでは、このステートメントは発行されません。

SET CURRENT IMPLICIT XMLPARSE OPTION ステートメントは、妥当性検査のない暗黙的な構文解析の際に空白文字を除去するか保存するかを制御する CURRENT IMPLICIT XMLPARSE OPTION 特殊レジスターを設定します。

CurrentImplicitXMLParseOption は、XMLPARSE 機能による明示的な構文解析には影響を与えません。

CurrentImplicitXMLParseOption では、次の設定値がサポートされます。

- STRIP WHITESPACE - XML 文書が暗黙的に構文解析されるとき、空白文字は除去されます。これはデフォルト設定です。
- PRESERVE WHITESPACE - XML 文書が暗黙的に構文解析されるとき、空白文字は保存されます。

CurrentMaintainedTableTypesForOpt CLI/ODBC 構成キーワード

CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターの値を設定します。

db2cli.ini キーワード構文:

CurrentMaintainedTableTypesForOpt = **ALL** | **FEDERATED_TOOL** | **NONE** | **SYSTEM** | **USER** | **<リスト>**

デフォルト設定:

照会の最適化において、システムの管理するリフレッシュ据え置きマテリアライズ照会表が考慮されます。

使用上の注意:

このキーワードは、 CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターのデフォルト値を定義します。この特殊レジスターの値は、照会の最適化において考慮される表の種類に影響します。サポートされている ALL、FEDERATED_TOOL、NONE、SYSTEM、または USER の設定値については、 SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION SQL ステートメントの説明を参照してください。 <リスト> オプションは、サポートされている複数の設定値の組み合わせを表しています。しかし、ALL と NONE については、他の値と組み合わせて指定することはできません。また、同じ値を複数回指定することもできません。リストの中では、値と値の間をコンマで区切ってください。例えば、

```
CurrentMaintainedTableTypesForOpt = SYSTEM,USER
```

CURRENTOPTIMIZATIONPROFILE CLI/ODBC 構成キーワード

接続成功時に SET CURRENT OPTIMIZATION ステートメントで使用される最適化プロファイルを指定します。

db2cli.ini キーワード構文:

CURRENTOPTIMIZATIONPROFILE =NULL|*optimization-profile-name*

デフォルト設定:

NULL

使用上の注意:

NULL
レジスターを NULL 値に設定します。

optimization-profile-name
CURRENT OPTIMIZATION PROFILE 特殊レジスターを最適化プロファイルの名前に設定します。optimization-profile-name が修飾されていない場合、デフォルトのスキーマ修飾が適用されます。

例

DB2CLI.INI ファイルに以下の項目がある場合、『Rochester』 データベースへの各接続が正常に実行された後、CLI クライアントはコマンド SET CURRENT OPTIMIZATION PROFILE = '"Hamid"."RochesterProfile"' を発行します。

```
[Rochester]
CURRENTOPTIMIZATIONPROFILE='"Hamid"."RochesterProfile"'
```

この例では、最適化プロファイル名に小文字が含まれるため、最適化プロファイル名は引用符で区切られています。

CurrentPackagePath CLI/ODBC 構成キーワード

すべての接続の後に、'SET CURRENT PACKAGE PATH = *schema1, schema2, ...*' を出します。

db2cli.ini キーワード構文:

CurrentPackagePath = *schema1, schema2, ...*

デフォルト設定:

節は付加されません。

同等の接続属性:

SQL_ATTR_CURRENT_PACKAGE_PATH

使用上の注意:

このオプションを設定すると、データベースに対するすべての接続の後にコマンド "SET CURRENT PACKAGE PATH = *schema1, schema2, ...*" が発行されます。この設定値は、他のスキーマからのパッケージが存在するときに検索されるスキーマ名 (コレクション ID) のリストを指定します。

このキーワードは、 CLI アプリケーションではなく ODBC 静的処理アプリケーションで使用することに最適です。

CurrentPackageSet CLI/ODBC 構成キーワード

接続するたびに SET CURRENT PACKAGESET ステートメントを発行します。

db2cli.ini キーワード構文:

CurrentPackageSet = スキーマ名

デフォルト設定:

節は付加されません。

同等の接続属性:

SQL_ATTR_CURRENT_PACKAGE_SET

使用上の注意:

このオプションは、データベースに接続するたびに、その後 SET CURRENT PACKAGESET SQL ステートメントを CurrentPackageSet 値とともに発行します。デフォルトでは、この節は付加されません。

SET CURRENT PACKAGESET SQL ステートメントは、後続の SQL ステートメントのために使用するパッケージの選択に使用されるスキーマ名 (コレクション ID) を設定します。

CLI/ODBC アプリケーションは、動的 SQL ステートメントを発行します。このオプションを使用すると、これらのステートメントの実行に使用される特権を制御することができます。

- CLI/ODBC アプリケーションから SQL ステートメントを実行するときに使用するスキーマを選択します。
- スキーマ内のオブジェクトに必要な特権があることを確認してから、それに従って再バインドします。

- CurrentPackageSet オプションをこのスキーマに設定します。

CLI/ODBC アプリケーションからの SQL ステートメントは、指定されたスキーマの下で実行され、そこで定義された特権を使用します。

次のパッケージ・セット名は予約済みです。
NULLID、NULLIDR1、NULLIDRA。

Reopt および CurrentPackageSet の両方のキーワードが指定されている場合には、CurrentPackageSet が優先します。

CurrentRefreshAge CLI/ODBC 構成キーワード

CURRENT REFRESH AGE 特殊レジスターの値を設定します。

db2cli.ini キーワード構文:

CurrentRefreshAge = **0** | **ANY** | **正整数**

デフォルト設定:

REFRESH IMMEDIATE で定義されたマテリアライズ照会表だけを使って照会処理を最適化できます。

使用上の注意:

このキーワードを設定すると、CURRENT REFRESH AGE 特殊レジスターの値が設定されます。

CurrentSQLID CLI/ODBC 構成キーワード

接続成功時に DBMS に送信される SET CURRENT SQLID ステートメントで使用される ID を指定します。

db2cli.ini キーワード構文:

CurrentSQLID = *current_sqlid*

デフォルト設定:

ステートメントは発行されません。

次の場合にのみ適用可能:

SET CURRENT SQLID がサポートされている DB2 DBMS に接続する。

使用上の注意:

このオプションが設定されている場合、接続成功時には SET CURRENT SQLID ステートメントが DBMS に送信されます。これにより、エンド・ユーザーおよびアプリケーションは、 SQL オブジェクトをスキーマ名で限定せずに指定することができます。

CurrentSchema CLI/ODBC 構成キーワード

接続成功時に SET CURRENT SCHEMA ステートメントで使用されるスキーマを指定します。

db2cli.ini キーワード構文:

CurrentSchema = スキーマ名

デフォルト設定:

ステートメントは発行されません。

使用上の注意:

このオプションが設定されている場合、接続成功時には SET CURRENT SCHEMA ステートメントが DBMS に送信されます。これにより、エンド・ユーザーまたはアプリケーションは、 SQL オブジェクトをスキーマ名で限定せずに指定することができます。

CursorHold CLI/ODBC 構成キーワード

トランザクション完了のオープン・カーソルへの影響を制御します。

db2cli.ini キーワード構文:

CursorHold = **1** | **0**

デフォルト設定:

選択 -- カーソルは破棄されません。

同等のステートメント属性:

SQL_ATTR_CURSOR_HOLD

使用上の注意:

このオプションは、トランザクション完了のオープン・カーソルへの影響を制御します。

- 1 = SQL_CURSOR_HOLD_ON、カーソルはトランザクションがコミットされても破棄されません (デフォルト)。
- 0 = SQL_CURSOR_HOLD_OFF、カーソルはトランザクションがコミットされると破棄されます。

注: カーソルはトランザクションがロールバックされるたびにクローズされます。

このオプションは、 SQL_CURSOR_COMMIT_BEHAVIOR または SQL_CURSOR_ROLLBACK_BEHAVIOR と一緒に呼び出されると、 `SQLGetInfo()` によって戻される結果に影響します。 保留カーソルがサポートされていない DB2 Server for VSE & VM への接続の場合、 CursorHold の値は無視されます。

このオプションを使用して、パフォーマンスを調整することができます。アプリケーションが以下にあてはまることが確実である場合は、SQL_CURSOR_HOLD_OFF (0) に設定できます。

1. `SQLGetInfo()` によって戻される SQL_CURSOR_COMMIT_BEHAVIOR または SQL_CURSOR_ROLLBACK_BEHAVIOR 情報に依存した動作を行わない。
2. あるトランザクションから次のトランザクションまでカーソルを保持する必要がない。

トランザクションの終了後にリソースを保守する必要がなくなるため、CursorHold が無効の状態で DBMS はより効果的に稼働します。

CursorTypes CLI/ODBC 構成キーワード

可能なカーソル・タイプを指定します。

db2cli.ini キーワード構文:

CursorTypes = **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7**

デフォルト設定:

サーバーでサポートされているなら、前方スクロール、静的、キー・セット主導、および動的の各カーソルがサポートされます。

使用上の注意:

CursorTypes キーワードは、アプリケーションでどのタイプのカーソルを開くことができるかを指定するためのビット・マスクです。

- 0x0 - 前方スクロール (常に可能)
- 0x1 - 静的
- 0x2 - キー・セット主導
- 0x4 - 動的

例えば、次のようになります。

- アプリケーションで動的両方向スクロール・カーソルを開くことができないようにするには、CursorTypes に 3 を設定します。
- アプリケーションが順方向カーソルのみ開くことができるようにするには、CursorTypes を 0 に設定します。

このキーワードは、以下の CLI 関数の呼び出しにのみ影響します。

- `SQLBulkOperations()`
- `SQLExecDirect()`
- `SQLExecute()`
- `SQLFetchScroll()`
- `SQLPrepare()`
- `SQLSetPos()`

DB2Degree CLI/ODBC 構成キーワード

SQL ステートメントの実行について並列処理の度合いを設定します。

db2cli.ini キーワード構文:

DB2Degree = **0** | **1 から 32767 までの整数値** | **ANY**

デフォルト設定:

SET CURRENT DEGREE ステートメントは発行されません。

次の場合にのみ適用可能:

クラスター・データベース・システムに接続する。

使用上の注意:

指定された値が 0 (デフォルト) 以外のものである場合、 CLI は接続成功後に次の SQL ステートメントを発行します。

```
SET CURRENT DEGREE value
```

これにより、SQL ステートメントの実行について並列処理の度合いが指定されます。 ANY を指定するとデータベース・マネージャーによって並列処理の度合いが決定されます。

DB2Explain CLI/ODBC 構成キーワード

Explain スナップショット、Explain 表、またはその両方の情報のうちどれがサーバーによって生成されるかを決定します。

db2cli.ini キーワード構文:

DB2Explain = **0** | **1** | **2** | **3**

デフォルト設定:

Explain スナップショットも Explain 表情報もサーバーによって生成されません。

同等の接続属性:

SQL_ATTR_DB2EXPLAIN

使用上の注意:

このキーワードは、Explain スナップショット、Explain 表、またはその両方の情報のうちどれがサーバーによって生成されるかを決定します。

- 0 = 両方をオフにします (デフォルト)

  'SET CURRENT EXPLAIN SNAPSHOT=NO' および 'SET CURRENT EXPLAIN MODE=NO' ステートメントがサーバーに送信され、Explain スナップショットおよび Explain 表情報キャプチャー機能の両方を無効にします。
- 1 = Explain スナップショット機能のみをオンにします

  'SET CURRENT EXPLAIN SNAPSHOT=YES' および 'SET CURRENT EXPLAIN MODE=NO' ステートメントがサーバーに送信され、Explain スナップショット機能を有効にして、Explain 表情報キャプチャー機能を無効にします。
- 2 = EXPLAIN 表情報キャプチャー機能のみをオンにします

  'SET CURRENT EXPLAIN MODE=YES' および 'SET CURRENT EXPLAIN SNAPSHOT=NO' がサーバーに送信され、Explain 表情報キャプチャー機能を有効にして、Explain スナップショット機能を無効にします。
- 3 = 両方をオンにします

  'SET CURRENT EXPLAIN MODE=YES' および 'SET CURRENT EXPLAIN SNAPSHOT=YES' がサーバーに送信され、Explain スナップショットおよび Explain 表情報キャプチャー機能の両方を有効にします。

EXPLAIN 情報は EXPLAIN 表に挿入されます。 EXPLAIN 情報を生成するには、EXPLAIN 表が作成されていなければなりません。現在の許可 ID に、EXPLAIN 表の INSERT 特権が必要です。

注釈

バージョン 9.7 フィックスパック 3 以降、使用可能なのは EXPLAIN モード情報のみであるため、DB2 for z/OS サーバーがサポートしている **DB2Explain** の設定値は 0 (オフ) と 2 (表) だけです。**DB2Explain** キーワードを、サポート対象になっていないデータ・サーバーに設定しようとすると、アプリケーションは「CLI0150E ドライバーが使用できません」というエラーを受け取ります。

DB2NETNamedParam CLI/ODBC 構成キーワード

名前付きパラメーターが DB2 .NET アプリケーションによって使用されるかどうかを指定します。

db2cli.ini キーワード構文:

DB2NETNamedParam = **0** | **1**

デフォルト設定:

IBM Data Server Provider for .NET は、SQL ステートメント内で、名前付きパラメーターをパラメーターとして認識しますが、位置パラメーターを無視します。

使用上の注意:

デフォルトでは、IBM Data Server Provider for .NET は SQL ステートメント内で形式が「@<paramname>」のトークンを名前付きパラメーターとして処理し、位置パラメーターは無視します。位置パラメーターは、「?」文字、またはコロンの後に名前が続く形 (*:name*) で指定されます。

以下の照会は、名前付きパラメーターを含む照会の例です。

```
SELECT * FROM T1 WHERE C1 = @param1
```

次は、位置パラメーターを含む照会の例です。

```
SELECT * FROM T1 WHERE C1 = ?
```

0 を指定すると、位置パラメーターだけが SQL ステートメント内でパラメーターとして認識されることを示します。この設定は、名前付きパラメーターの処理に必要なリソースを削減することによって、アプリケーションのパフォーマンスを改善することができます。

DB2Optimization CLI/ODBC 構成キーワード

照会最適化レベルを設定します。

db2cli.ini キーワード構文:

DB2Optimization = *0 から 9 までの整数値*

デフォルト設定:

SET CURRENT QUERY OPTIMIZATION ステートメントは発行されません。

使用上の注意:

このオプションが設定されると、CLI は接続成功後に次の SQL ステートメントを発行します。

```
SET CURRENT QUERY OPTIMIZATION positive number
```

これにより、オプティマイザーが SQL 照会を操作する照会最適化レベルが指定されます。

DBAlias CLI/ODBC 構成キーワード

8 文字より多いデータ・ソース名 (DSN) のデータベース別名を指定します。

db2cli.ini キーワード構文:

DBAlias = *dbalias*

デフォルト設定:

ODBC データ・ソース名として DB2 データベース別名を使用します。

使用上の注意:

DSN は、大括弧で囲まれた名前です。これは、db2cli.ini ファイルのセクション・ヘッダーを示します。一般に、このセクション・ヘッダーは最大 8 バイトの長さのデータベース別名です。これよりも長く意味のある名前を使用するには、その名前をセクション・ヘッダーに入れ、このキーワード値を **CATALOG** コマンドで使用するデータベース別名に設定します。以下はその例です。

```
; The much longer name maps to an 8 single byte character dbalias
[MyMeaningfulName]
DBAlias=DB2DBT10
```

実際のデータベース別名が DB2DBT10 でも、接続時にデータ・ソースの名前として [MyMeaningfulName] を指定することができます。

db2cli.ini ファイルでデータベース・キーワードと共に **DBAlias** キーワードに値を指定した場合、データベースに接続しようとするアプリケーションや、この値に一致する DSN は、エラーを受け取りません。

DBName CLI/ODBC 構成キーワード

アプリケーションが z/OS または OS/390 表情報の照会に要する時間を削減するために、データベース名を指定します。

db2cli.ini キーワード構文:

DBName = *dbname*

デフォルト設定:

DBNAME 列でフィルターをかけません。

次の場合にのみ適用可能:

DB2 for z/OS and OS/390 に接続する。

使用上の注意:

このオプションは、DB2 for z/OS および OS/390 に接続しているとき、およびアプリケーションによって (基本) 表カタログ情報が要求された場合にのみ使用されま

す。 z/OS または OS/390 サブシステムに存在する表の数が多い場合、アプリケーションが表情報の照会に要する時間を削減し、アプリケーションによってリストされる表の数を減らすために、 *dbname* を指定できます。

このオプションが設定されると、 IN DATABASE *dbname* ステートメントは CREATE TABLE などのさまざまなステートメントに付加されます。

この値は z/OS または OS/390 システム・カタログ表の DBNAME 列にマップします。値が指定されていない場合、あるいはビュー、シノニム、システム表、または別名が TableType を介して指定されている場合も、表情報のみが制限されます。ビュー、別名、およびシノニムは、DBName で制限されません。これを SchemaList および TableType とともに使用して、情報が戻される表の数をさらに制限することができます。

DSN CLI/ODBC 構成キーワード

SQLDataSources によって戻されるデータ・ソースの名前、または SQLDriverConnect のデータ・ソース・ダイアログ・ボックスによって戻されるデータ・ソースの名前を設定します。

db2cli.ini キーワード構文:

このキーワードは db2cli.ini ファイル内では設定できません。

SQLDriverConnect 内の接続ストリングで、このキーワードの値を以下のように指定できます。

```
DSN = データベース名
```

Database CLI/ODBC 構成キーワード

ファイル DSN を使用するときは、接続するサーバーのデータベースを指定します。

db2cli.ini キーワード構文:

Database = データベース名

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

ファイル DSN を使用するときは、このオプションを使用して、接続するサーバーのデータベースを指定する必要があります。この値はクライアントで指定されるデータベース別名とは関係ありません。これは、サーバーのデータベース名自体に設定される必要があります。

この設定は、Protocol オプションが TCPIP に設定されているときにのみ考慮されます。

DateTimeStringFormat CLI/ODBC 構成キーワード

日付または時間データを文字型の列に挿入する際に使用する形式を指定します。

db2cli.ini キーワード構文:

DateTimeStringFormat = **JIS** | **ISO** | **EUR** | **USA**

デフォルト設定:

日付または時間データを文字型の列に挿入する際、JIS 形式が使用されます。

使用上の注意:

DateTimeStringFormat キーワードは、日付または時間データを文字型の列に挿入する際の形式を制御します。この設定は、
SQL_C_TYPE_DATE、SQL_C_TYPE_TIME、 SQL_C_TYPE_TIMESTAMP、または SQL_C_TIMESTAMP_EXT のデータを、以下のタイプの列に挿入する方法に影響します。

- SQL_CHAR
- SQL_VARCHAR
- SQL_LONGVARCHAR
- SQL_CLOB

このキーワードは、日付または時刻の列から取り出されて文字ストリングに入れられるデータの形式にも影響します。例えば、SQL_TYPE_TIMESTAMP 列からデータを取り出して SQL_C_CHAR ストリングに入れる場合、このキーワードの設定が影響します。

表 159. 設定値

形式	日付	時刻	タイム・スタンプ
JIS	yyyy-mm-dd	hh:mm:ss	yyyy-mm-dd hh:mm:ss.fffffffffff
ISO	yyyy-mm-dd	hh.mm.ss	yyyy-mm-dd-hh.mm.ss.fffffffffff
EUR	dd.mm.yyyy	hh.mm.ss	yyyy-mm-dd hh:mm:ss.fffffffffff*
USA	mm/dd/yyyy	hh:mm AM または PM	yyyy-mm-dd hh:mm:ss.fffffffffff*
* EUR または USA が指定された場合、タイム・スタンプはデフォルトの形式になります。デフォルトの形式は JIS です。			

DecimalFloatRoundingMode CLI/ODBC 構成キーワード

DECFLOAT SQL タイプをサポートするサーバーを扱う場合の丸めモードを設定します。

db2cli.ini キーワード構文:

DecimalFloatRoundingMode = **0** | 1 | 2 | 3 | 4

デフォルト設定:

0 (Half even 丸めモード)

同等の接続属性:

SQL_ATTR_DECFLOAT_ROUNDING_MODE

使用上の注意:

10 進数浮動小数点丸めモードは、DECFLOAT の変数または列に入っている値の桁数が DECFLOAT データ・タイプで許可される桁数を超えている場合に、どのタイプの丸めを使用するかを決定します。このようなことは、別のタイプから挿入、更新、選択、変換する際、あるいは数学演算の結果として生じることがあります。

SQL_ATTR_DECFLOAT_ROUNDING_MODE の値は、新しい接続の接続属性によって他のモードが指定されていない場合に、その新しい接続で使用される 10 進数浮動小数点丸めモードを決定します。指定されたどの接続に対しても、CLI および DB2 の両方が、その接続の一部として開始されるすべてのアクションで、同じ 10 進数浮動小数点丸めモードを使用します。

アプリケーションが DB2 Database for Linux, UNIX, and Windows バージョン 9.5 サーバーに接続する場合は、データベース・クライアントの 10 進数浮動小数点丸めモードを、サーバーで設定されているのと同じモードに設定する必要があります。クライアントの 10 進数浮動小数点丸めモードを、データベース・サーバーで設定されている 10 進数浮動小数点丸めモードとは異なる値に設定していると、接続時にデータベース・サーバーから SQL0713N が戻されます。

これらの 10 進数浮動小数点丸めモードに対応する設定値は、次のとおり。

- 0 = Half even (デフォルト)
- 1 = Half up
- 2 = Down
- 3 = Ceiling
- 4 = Floor

各モードは、次のとおりです。

Half even (デフォルト)
このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値に最も近似の数値を使用します。 2 つの数値がほぼ同じ程度の近似値の場合には、偶数であるものを使用します。このモードは、大量のデータに対して、生じる丸め誤差を最も小さくします。

Half up
このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値に最も近似の数値を使用します。 2 つの数値がほぼ同じ程度の近似値の場合には、元の値よりも大きいものを使用します。

Down このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値に最も近似で、その絶対値が元の値の絶対値よりも大きくない数値を使用します。これは、ゼロに向かっての丸め、または負の値の場合の Ceiling の使用、および正の値の場合の Floor の使用とみなすことができます。

Ceiling
このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値より大か等しい数値で最小のものを使用します。

Floor このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値より小か等しい数値で最大のものを使用します。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

DeferredPrepare CLI/ODBC 構成キーワード

PREPARE 要求を、対応する実行要求と結合することにより、ネットワークの流れを最小化します。

db2cli.ini キーワード構文:

DeferredPrepare = **0** | **1**

デフォルト設定:

準備要求は実行要求が送信されるまで実行されません。

同等のステートメント属性:

SQL_ATTR_DEFERRED_PREPARE

使用上の注意:

対応する実行要求が発行されるまで、PREPARE 要求の送信を据え置きます。その後、ネットワーク・フローを最小化しパフォーマンスを改善するため、 2 つの要求が 2 つではなく 1 つのコマンド/応答のフローに結合されます。

- 0 = SQL_DEFERRED_PREPARE_OFF。 PREPARE 要求は、発行された時点で実行されます。
- 1 = SQL_DEFERRED_PREPARE_ON (デフォルト)。対応する実行要求が発行されるまで、PREPARE 要求の実行は据え置かれます。

  ターゲット DBMS が据え置き準備をサポートしていない場合、クライアントはその接続のための据え置き準備を無効にします。

注: 据え置き準備を有効にすると、通常は SQLCA の PREPARE ステートメントの SQLERRD(3) と SQLERRD(4) に戻される行およびコスト見積もりが、ゼロになる可能性があります。これは、これらの値を使用して SQL ステートメントを続行するかどうかを決定するユーザーにとっては重要です。

DescribeCall CLI/ODBC 構成キーワード

ストアード・プロシージャー引数がいつ記述 (DESCRIBE) されるかを判別します。

db2cli.ini キーワード構文:

DescribeCall = **1** | **-1**

デフォルト設定:

CALL ステートメントを準備する際に、DB2 CLI はストアード・プロシージャー引数記述情報を要求しません。

同等の接続属性:

SQL_ATTR_DESCRIBE_CALL

使用上の注意:

デフォルトでは、CALL ステートメントを準備する際に、CLI は入力パラメーター記述情報を要求しません。アプリケーションがパラメーターをステートメントに正常にバインドした場合には、この記述情報は不要になるため、記述情報を要求しないとパフォーマンスが改善されます。

オプションの値は、次のとおりです。

- 1 = SQL_DESCRIBE_CALL_BEFORE。 CLI は常に、サーバーからの記述情報を要求し、アプリケーションによって提供されたバインディング情報を無視します。 DescribeCall を 1 に設定すると、さらに DeferredPrepare が 0 に設定されます。これは、記述情報が動的 SQL ステートメントについても要求されることを意味します。 DeferredPrepare を 0 に設定しても、DescribeCall は 1 に設定されないことに注意してください。
- -1 = SQL_DESCRIBE_CALL_DEFAULT (デフォルト)。CLI は、サーバーからの記述情報を要求せず、アプリケーションによって提供されたバインディング情報を使用します。 CALL ステートメントの実行が失敗した場合は、CLI エラー・リカバリー・ロジックが、サーバーからの入力パラメーター記述情報を要求し、CALL ステートメントを再び発行します。

DescribeInputOnPrepare CLI/ODBC 構成キーワード

SQL ステートメントを準備するときに、記述情報の要求を有効または無効にします。

db2cli.ini キーワード構文:

DescribeInputOnPrepare = **0** | **1**

デフォルト設定:

SQL ステートメントの準備中に、記述情報を要求しません。

使用上の注意:

デフォルトでは、SQL ステートメントを準備する際に、 CLI は入力パラメーター記述情報を要求しません。アプリケーションがパラメーターをステートメントに正常にバインドした場合には、この記述情報は不要になるため、記述情報を要求しないとパフォーマンスが改善されます。しかし、パラメーターが正常にバインドされなかった場合には、ステートメントの実行が失敗し、CLI エラー・リカバリー論理によって入力パラメーター記述情報が要求されることになります。結果として、記述情報が準備の際に要求された場合と比べると、サーバー要求が増え、パフォーマンスは低下します。 DescribeInputOnPrepare を 1 に設定すると、入力記述情報は準備の際に要求されるようになります。このように設定すると、バインド・エラーからリカバリーを CLI 再試行論理にかなり依存しているアプリケーションのパフォーマンスが改善できる場合があります。

DescribeOutputLevel CLI/ODBC 構成キーワード

準備 (PREPARE) 要求または記述 (DESCRIBE) 要求中に CLI ドライバーによって要求される、出力列記述情報のレベルを設定します。

db2cli.ini キーワード構文:

DescribeOutputLevel = 0 | 1 | 2 | 3

デフォルト設定:

400 ページの表 160 のレベル 2 にリストされた記述情報を要求します。

同等の接続属性:

SQL_ATTR_DESCRIBE_OUTPUT_LEVEL

使用上の注意:

このキーワードは、準備要求または記述要求で CLI ドライバーが要求する情報量を制御します。デフォルトでは、サーバーが記述要求を受け取ると、結果セット列に 400 ページの表 160 のレベル 2 に含まれる情報を戻します。ただし、アプリケーションがこの情報のすべてを必要としないことも、あるいは追加情報を必要とすることもあります。 **DescribeOutputLevel** キーワードを、クライアント・アプリケーションの要件に合ったレベルに設定すると、パフォーマンスが改善される可能性があります。なぜなら、クライアントとサーバー間で転送される記述データが、アプリケーションが必要とする最小量に制限されるためです。 **DescribeOutputLevel** 設定値が低すぎると、アプリケーションの機能に影響を与えることがあります (アプリケーションの要件によって異なる)。この場合、記述情報を検索する CLI 関数は失敗しないかもしれませんが、戻される情報は不完全である可能性があります。 **DescribeOutputLevel** のサポートされる設定値は、次のとおりです。

- 0 - 記述情報は、クライアント・アプリケーションに戻されません。
- 1 - レベル 1 に分類される記述情報 (400 ページの表 160 を参照) が、クライアント・アプリケーションに戻されます。
- 2 - (デフォルト) レベル 2 に分類される記述情報 (400 ページの表 160 を参照) が、クライアント・アプリケーションに戻されます。
- 3 - レベル 3 に分類される記述情報 (400 ページの表 160 を参照) が、クライアント・アプリケーションに戻されます。

以下の表は、サーバーが準備要求または記述要求を受け取ったときに戻す、記述情報を形成するフィールドをリストします。これらのフィールドは各レベルにグループ化され、**DescribeOutputLevel** CLI/ODBC 構成キーワードが、CLI ドライバーの要求する記述情報のレベルを制御します。

注:

1. 必ずしもすべての記述情報のレベルが、すべての DB2 サーバーによってサポートされるとは限りません。記述情報のすべてのレベルがサポートされるのは、次の DB2 サーバーにおいてです。DB2 for Linux, UNIX, and Windows バージョン 8 以降、DB2 for z/OS バージョン 8 以降、および DB2 for i5/OS® バージョン 5 リリース 3 以降。他のすべての DB2 サーバーは、**DescribeOutputLevel** に対して 2 または 0 の設定値のみサポートします。

2. デフォルトの動作では、アプリケーションがデフォルトのレベル 2 を使用して最初に検索されなかった記述情報を要求するとき、CLI はレベル 3 にプロモートされます。その結果、サーバーに対する 2 つのネットワーク・フローが生じることがあります。アプリケーションがこのキーワードを使用して記述レベルを明示的に設定した場合は、プロモーションは発生されません。そのため、キーワードを使用して記述レベルを 2 に設定した場合には、アプリケーションが拡張情報を要求する場合でも CLI はレベル 3 にプロモートされません。

表 160. 記述情報のレベル

レベル 1	レベル 2	レベル 3
SQL_DESC_COUNT SQL_COLUMN_COUNT SQL_DESC_TYPE SQL_DESC_CONCISE_TYPE SQL_COLUMN_LENGTH SQL_DESC_OCTET_LENGTH SQL_DESC_LENGTH SQL_DESC_PRECISION SQL_COLUMN_PRECISION SQL_DESC_SCALE SQL_COLUMN_SCALE SQL_DESC_DISPLAY_SIZE SQL_DESC_NULLABLE SQL_COLUMN_NULLABLE SQL_DESC_UNSIGNED SQL_DESC_SEARCHABLE SQL_DESC_LITERAL_SUFFIX SQL_DESC_LITERAL_PREFIX SQL_DESC_CASE_SENSITIVE SQL_DESC_FIXED_PREC_SCALE	all fields of level 1 and: SQL_DESC_NAME SQL_DESC_LABEL SQL_COLUMN_NAME SQL_DESC_UNNAMED SQL_DESC_TYPE_NAME SQL_DESC_DISTINCT_TYPE SQL_DESC_REFERENCE_TYPE SQL_DESC_STRUCTURED_TYPE SQL_DESC_USER_TYPE SQL_DESC_LOCAL_TYPE_NAME SQL_DESC_USER_DEFINED_ TYPE_CODE	all fields of levels 1 and 2 and: SQL_DESC_BASE_COLUMN_NAME SQL_DESC_UPDATABLE SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_SCHEMA_NAME SQL_DESC_CATALOG_NAME SQL_DESC_TABLE_NAME SQL_DESC_BASE_TABLE_NAME

DescribeParam CLI/ODBC 構成キーワード

SQLDescribeParam() 関数を有効または無効にします。

db2cli.ini キーワード構文:

DescribeParam = **0** | **1**

デフォルト設定:

SQLDescribeParam() 関数は有効になっています。

使用上の注意:

1 (デフォルト) に設定すると、SQLDescribeParam() は有効となり、SQLGetFunctions() はサポートされているものとして SQLDescribeParam() を戻します。

0 に設定すると、SQLDescribeParam() は無効となります。 SQLDescribeParam() が呼び出されると、CLI0150E が戻されます。SQLGetFunctions() はサポートされていないものとして SQLDescribeParam() を戻します。

DiagLevel CLI/ODBC 構成キーワード

診断レベルを設定します。

db2cli.ini キーワード構文:

DiagLevel = **0** | **1** | **2** | **3** | **4**

デフォルト設定:

3

使用上の注意:

これは、db2cli.ini ファイルの [COMMON] セクションでのみ設定できます。

これは、プロセス全体のための環境ハンドルの割り振り時にのみ適用されます。

これは、データベース・マネージャーのパラメーター DIAGLEVEL と同等です。

DiagPath CLI/ODBC 構成キーワード

db2diag ログ・ファイルのパスを設定します。

db2cli.ini キーワード構文:

DiagPath = *既存のディレクトリー*

デフォルト設定:

デフォルト値は、UNIX および Linux オペレーティング・システムでは db2dump ディレクトリー、Windows オペレーティング・システムでは db2 ディレクトリーです。

使用上の注意:

これは、db2cli.ini ファイルの [COMMON] セクションでのみ設定できます。

これは、データベース・マネージャーのパラメーター DIAGPATH と同等です。

DisableKeysetCursor CLI/ODBC 構成キーワード

キー・セット主導両方向スクロール・カーソルを無効にします。

db2cli.ini キーワード構文:

DisableKeysetCursor = **0** | **1**

デフォルト設定:

キー・セット主導両方向スクロール・カーソルは要求時に戻されます。

使用上の注意:

1 に設定した場合、このキーワードは、アプリケーションがキー・セット主導両方向スクロール・カーソルを要求した場合でも、CLI ドライバーがアプリケーションに対し静的カーソルを戻すように強制します。デフォルトの設定値 (0) により、キ

ーセット・ドリブン・カーソルは、アプリケーションの要求時に戻されます。このキーワードは、両方向スクロール・カーソルがサポートされるまで、動作のリストアに使用できます。

DisableMultiThread CLI/ODBC 構成キーワード

マルチスレッド化を無効にします。

db2cli.ini キーワード構文:

DisableMultiThread = **0** | **1**

デフォルト設定:

マルチスレッド化は有効です。

使用上の注意:

CLI/ODBC ドライバーは複数の並行スレッドをサポートできます。

このオプションは、マルチスレッド・サポートを有効または無効にするために使用されます。
- 0 = マルチスレッド化を有効にします (デフォルト)。
- 1 = マルチスレッド化を無効にします。

マルチスレッド化が無効の場合、すべてのスレッドのすべての呼び出しが処理レベルでシリアライズされます。動作をシリアライズする必要のあるマルチスレッド化アプリケーションのこの設定を使用してください。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

DisableUnicode CLI/ODBC 構成キーワード

基礎 Unicode サポートを無効にします。

db2cli.ini キーワード構文:

DisableUnicode = **<設定しない>** | **0** | **1**

デフォルト設定:

Unicode サポートは有効です。

使用上の注意:

Unicode サポートが有効の場合で、Unicode アプリケーションにより呼び出された場合は、CLI は可能なクライアント・コード・ページで最良のものを使用してデータベースへの接続を試行し、コード・ページの変換による不要なデータの損失がないことを確認します。これにより、コード・ページの変更時に接続時間が増加するか、またはこのサポートが追加されるまでは発生しなかったクライアントのコード・ページの変換が発生する場合があります。

アプリケーションが Unicode の場合 (SQL_ATTR_ANSI_APP 接続属性が SQL_AA_FALSE に設定されているか、SQLConnectW() を使用して接続された場合)、**DisableUnicode** キーワードを使用して 3 つの異なる接続動作を設定できます。

- **DisableUnicode** が db2cli.ini ファイルで設定されていない場合: ターゲット・データベースが Unicode をサポートしていれば、CLI は Unicode コード・ページ (1208 および 1200) で接続します。ターゲット・データベースが Unicode をサポートしていなければ、 CLI はアプリケーションのコード・ページで接続します。
- **DisableUnicode**=0 が設定されている場合: CLI は、ターゲット・データベースが Unicode をサポートしているかどうかに関係なく、常に Unicode で接続します。
- **DisableUnicode**=1 が設定されている場合: CLI は、ターゲット・データベースが Unicode をサポートしているかどうかに関係なく、常にアプリケーションのコード・ページで接続します。

EnableNamedParameterSupport CLI/ODBC 構成キーワード

名前付きパラメーターの処理が有効かどうかを指定します。

db2cli.ini キーワード構文:

EnableNamedParameterSupport = **TRUE** | **FALSE**

デフォルト設定:

名前付きパラメーターのサポートはオフです (FALSE)。

同等の環境または接続属性:

なし

使用上の注意:

名前付きパラメーターの処理を行う場合、アプリケーションは、従来の名前なしパラメーター・マーカー (疑問符 (?) によって表される) に加えて、名前付きパラメーター (:name など) を使用できます。名前付きパラメーターのサポートを有効または無効にするために使用できるオプションは以下のとおりです。

- TRUE - 名前付きパラメーターの処理が有効です。DB2for Linux, UNIX, and Windowsバージョン 9.7 以降の場合、CLI は、ステートメント・テキストを処理のためにそのままサーバーに送信します。他のすべてのサーバーの場合、CLI は、名前付きパラメーターを疑問符 (?) で置き換えることによって、ステートメント・テキストを置き換えてから、そのステートメントを処理のためにサーバーに送信します。
- FALSE - 名前付きパラメーターの処理はオフです。CLI はパラメーター・マーカーを処理しません。

名前ごとにバインドする機能はサポートされていません。CLI は、有効なパラメーター・マーカーに一致するものすべてを処理し、それが疑問符 (?) によって表される標準のパラメーター・マーカーであるかのように扱います。

FET_BUF_SIZE CLI/ODBC 構成キーワード

データ・フローを最適化するためにデフォルトの照会ブロック・サイズを指定します。

db2cli.ini キーワード構文:

FET_BUF_SIZE = **64K** | **96K** | **128K** | **160K** | **192K** | **224K** | **256K**

デフォルト設定:

FET_BUF_SIZE = 64K

同等の接続属性:

SQL_ATTR_FET_BUF_SIZE

使用上の注意:

CLI で許可されている照会ブロック・サイズは 32K の倍数のみ (つまり、64K、96K、128K、160K、192K、224K、および 256K) です。CLI アプリケーションでは、 64K から 256K の範囲内にあるその他の値は、次に一番近い 32K 境界に切り上げられます。

アプリケーションで、 SQLGetConnectAttr() を使用して、この属性に設定された値を取得することが可能です。アプリケーションでいずれの値も設定されていない場合、デフォルトの照会ブロック・サイズが返されます。

FileDSN CLI/ODBC 構成キーワード

データ・ソース用の接続ストリングを構築する基になる DSN ファイルを指定します。

db2cli.ini キーワード構文:

このキーワードは db2cli.ini ファイル内では設定できません。

SQLDriverConnect 内の接続ストリングで、このキーワードの値を以下のように指定できます。

```
FileDSN = file name
```

FloatPrecRadix CLI/ODBC 構成キーワード

浮動小数点タイプの NUM_PREC_RADIX 値を 2 または 10 に強制的に指定します。

db2cli.ini キーワード構文:

FloatPrecRadix = **2** | **10**

デフォルト設定:

浮動小数点タイプが 10 ではなく 2 を基数としている場合に、浮動小数点タイプについて NUM_PREC_RADIX を 2 と報告します。

使用上の注意:

NUM_PREC_RADIX 値はデータ・タイプの基数を表します。浮動小数点数などのバイナリー数は 2 を基数とし、整数は 10 を基数とします。アプリケーションは最大桁数を表すために、 COLUMN_SIZE フィールドにあるすべての値を予期しますが、その場合に想定される NUM_PREC_RADIX 値は 10 です。ただし、浮動小数

点数タイプの場合、 NUM_PREC_RADIX は 2 であり、このとき COLUMN_SIZE はデータ・タイプの表記に最大桁数ではなくビット数を報告します。

FloatPrecRadix は、浮動小数点データ・タイプに対しては NUM_PREC_RADIX が 10 として報告されるように強制することができ、この場合 COLUMN_SIZE は最大桁数を報告します。

FloatPrecRadix キーワードは SQLColumns()、 SQLGetDescField() (SQL_DESC_NUM_PREC_RADIX フィールドの場合)、 SQLGetTypeInfo()、 SQLProcedureColumns()、および SQLSpecialColumns() に影響します。

GetDataLobNoTotal CLI/ODBC 構成キーワード

SQLGetData() により列データがすべて一度にフェッチされるのではなく、列データを指定したサイズ (バイト単位) の断片でフェッチされるようにします。

db2cli.ini キーワード構文:

GetDataLobNoTotal = *正整数*

使用上の注意:

SQLGetData() は、結果セットの現在行で 1 つの列のデータを取り出します。 SQLGetData() を最初に呼び出す際に、以下のタスクが行われます。

- データベース・サーバーからクライアントにすべてのデータがフェッチされます。これには、クライアントでデータ用のメモリーを割り振ることが必要となります
- そのデータにコード・ページ変換が適用されます (必要な場合)
- 変換データの長さの合計が計算されます
- 変換データの長さがクライアント・アプリケーションに返されます

データが大きい場合、クライアントでのデータ用メモリーの割り振りに失敗する場合があります。**GetDataLobNoTotal** キーワードを使用することにより、この潜在的なメモリー割り振りの問題を回避できます。

GetDataLobNoTotal キーワードを設定すると、SQLGetData() の最初の呼び出し時に、指定された列のデータは一括してフェッチされません。代わりに、SQLGetData() により、**GetDataLobNoTotal** の値で指定されたクライアント上のバッファーを満たすだけのデータがフェッチされ、サーバーからフェッチするデータがまだある場合には SQL_NO_TOTAL (-4) が返されます。すべてのデータをフェッチするために必要なだけ何度でも SQLGetData() を呼び出すことができます。すべてのデータをフェッチすると、SQLGetData() により SQL_SUCCESS および最後のデータ・チャンクのサイズが返されます。

GranteeList CLI/ODBC 構成キーワード

アプリケーションが表または列特権のリストを取得するときに戻される情報量を削減します。

db2cli.ini キーワード構文:

GranteeList = " '*userID1*', '*userID2*',... '*userIDn*' "

デフォルト設定:
結果にフィルターをかけません。

使用上の注意:

このオプションは、アプリケーションがデータベース内の表または表内の列の特権のリストを取得するときに戻される情報量を削減するために使用されます。指定された許可 ID のリストがフィルターとして使用されます。これらの *ID* に対して 付与された特権を持つ表または列のみが戻されます。

このオプションは、特権を付与された 1 つ以上の許可 ID を単一引用符で囲み、コンマで区切ったリストとして設定してください。ストリング全体が二重引用符で囲まれている必要もあります。例:

```
GranteeList=" 'USER1', 'USER2', 'USER8' "
```

この例では、アプリケーションが特定の表について特権のリストを取得する場合に、 *USER1*、*USER2*、または *USER8* に対して 付与された特権を持つ列のみが戻されます。

GrantorList CLI/ODBC 構成キーワード

アプリケーションが表または列特権のリストを取得するときに戻される情報量を削減します。

db2cli.ini キーワード構文:
GrantorList = " '*userID1*', '*userID2*',... '*userIDn*' "

デフォルト設定:
結果にフィルターをかけません。

使用上の注意:

このオプションは、アプリケーションがデータベース内の表または表内の列の特権のリストを取得するときに戻される情報量を削減するために使用されます。指定された許可 ID のリストがフィルターとして使用されます。これらの *ID* によって 付与された特権を持つ表または列のみが戻されます。

このオプションは、特権を付与された 1 つ以上の許可 ID を単一引用符で囲み、コンマで区切ったリストとして設定してください。ストリング全体が二重引用符で囲まれている必要もあります。例:

```
GrantorList=" 'USER1', 'USER2', 'USER8' "
```

この例では、アプリケーションが特定の表について特権のリストを取得する場合に、 *USER1*、*USER2*、または *USER8* によって 付与された特権を持つ列のみが戻されます。

Graphic CLI/ODBC 構成キーワード

CLI が、サポートされる SQL データ・タイプとして SQL_GRAPHIC (2 バイト文字) を戻すかどうか、および GRAPHIC 列の長さを報告する際にどんな単位を使用するかを指定します。

db2cli.ini キーワード構文:

Graphic = **0** | **1** | **2** | **3**

デフォルト設定:

SQL_GRAPHIC GRAPHIC データは、サポートされている SQL データ・タイプとしては戻されず、 GRAPHIC 列の長さは、その列の DBCS 文字の数の最大値になります。

使用上の注意:

Graphic キーワードは、 `SQLGetTypeInfo()` が呼び出された場合にサポートされる SQL データ・タイプとして SQL_GRAPHIC (2 バイト文字) データ・タイプが報告されるかどうか、また、出力引数として、または結果セットの一部として、長さまたは精度を戻すすべての CLI 関数において GRAPHIC 列の長さを報告するために使用される単位を指定します。

Graphic キーワードは、次のように設定します。

- 0 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとしては戻されず、 GRAPHIC 列の長さとして報告される値はその列の DBCS 文字の数の最大値になります。
- 1 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとして戻され、 GRAPHIC 列の長さとして報告される値はその列の DBCS 文字の数の最大値になります。
- 2 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとしては戻されず、 GRAPHIC 列の長さとして報告される値はその列のバイト数の最大値になります。
- 3 - SQL_GRAPHIC は、サポートされている SQL データ・タイプとしては戻され、 GRAPHIC 列の長さとして報告される値はその列のバイト数の最大値になります。

Hostname CLI/ODBC 構成キーワード

ファイル DSN 接続で、または DSN なし接続で使用されるサーバー・システムのホスト名または IP アドレスを指定します。

db2cli.ini キーワード構文:

Hostname = *ホスト名* | *IP アドレス*

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

このオプションは、このクライアント・マシンと DB2 が実行されているサーバーとの TCP/IP 接続に必要な属性を指定するために、 ServiceName オプションと共に使用します。これらの 2 つの値は、Protocol オプションが TCPIP に設定されているときにのみ使用されます。

サーバー・システムのホスト名または IP アドレスのどちらかを指定します。

IgnoreWarnList CLI/ODBC 構成キーワード

指定された sqlstate を無視します。

db2cli.ini キーワード構文:

IgnoreWarnList = 『'sqlstate1', 'sqlstate2', ...』

デフォルト設定:

警告は通常どおりに戻されます

使用上の注意:

まれに、アプリケーションが処理できない警告メッセージがあるが、すべての警告メッセージは無視しないという場合があります。このキーワードは、アプリケーションに渡さない警告を指示するために使用できます。データベース・マネージャーのすべての警告を無視する場合には、 IgnoreWarnings キーワードを使用してください。

sqlstate は、IgnoreWarnList と WarningList の両方に組み込まれている場合にはすべて無視されます。

それぞれの sqlstate は、大文字で指定し、単一引用符で囲み、コンマで区切る必要があります。ストリング全体が二重引用符で囲まれている必要もあります。例:

```
IgnoreWarnList="'01000', '01004','01504'"
```

IgnoreWarnings CLI/ODBC 構成キーワード

データベース・マネージャーの警告を無視します。

db2cli.ini キーワード構文:

IgnoreWarnings = **0** | **1**

デフォルト設定:

警告は通常どおりに戻されます。

使用上の注意:

まれなことではありますが、アプリケーションが警告メッセージを正しく処理しない場合があります。このキーワードを使用することにより、データベース・マネージャーからの警告のうち、アプリケーションに渡さないものを指定できます。可能な設定値は次のとおりです。

- 0 - 警告は通常の方法で報告されます (デフォルト)。

- 1 - データベース・マネージャーの警告は無視され、 SQL_SUCCESS が戻されます。一方、DB2 CLI/ODBC ドライバーからの警告は戻されます。通常の操作では、その多くが必要です。

このキーワードはそれだけでも使用できますが、WarningList CLI/ODBC 構成キーワードと共に使用することもできます。

Instance CLI/ODBC 構成キーワード

ファイル DSN 接続または DSN なし接続に関するローカル IPC 接続のインスタンス名を指定します。

db2cli.ini キーワード構文:

Instance = *インスタンス名*

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このキーワードを設定するときには、以下のオプションもまた設定する必要があります。

- Database
- Protocol=IPC

Interrupt CLI/ODBC 構成キーワード

割り込み処理モードを設定します。

db2cli.ini キーワード構文:

Interrupt = **0** | **1** | **2**

デフォルト設定:

1

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このオプションを設定するとき、以下のオプションもまた設定する必要があります。

- Database
- Protocol=IPC

キーワード値の意味は以下のとおりです。

0 処理の割り込みを使用不可にします (SQLCancel 呼び出しによって処理が割り込みされることはありません)。

1 割り込みがサポートされます (デフォルト)。このモードでは、サーバーが割り込みをサポートする場合、割り込みが送信されます。そうでない場合、接続がドロップされます。

INTERRUPT_ENABLED (DB2 Connect ゲートウェイ設定) および DB2 レジストリー変数 DB2CONNECT_DISCONNECT_ON_INTERRUPT の設定値は、Interrupt キーワードの設定値 1 よりも優先されます。

2 サーバーの割り込み能力にかかわらず、割り込みによって接続がドロップされます (SQLCancel は接続をドロップします)。

KRBPlugin CLI/ODBC 構成キーワード

ファイル DSN 接続または DSN なし接続でのクライアント側の認証に使用される Kerberos プラグイン・ライブラリーの名前を指定します。

db2cli.ini キーワード構文:

KRBPlugin = *プラグイン名*

デフォルト設定:

デフォルト値は、UNIX オペレーティング・システムではヌル、Windows オペレーティング・システムでは `IBMkrb5` です。

使用上の注意:

これは、特定のデータ・ソースに関する `db2cli.ini` ファイルの `[Data Source]` セクション内、または接続ストリング内で設定できます。

このパラメーターは、クライアント側の接続認証に使用される Kerberos プラグイン・ライブラリーの名前を指定します。このプラグインは、Kerberos 認証を使ってクライアントが認証されるときに使用されます。

KeepDynamic CLI/ODBC 構成キーワード

CLI アプリケーションが KEEPDYNAMIC 機能を利用できるかどうかを指定します。

db2cli.ini キーワード構文:

KeepDynamic = **0** | **1**

デフォルト設定:

CLI アプリケーションは KEEPDYNAMIC 機能を利用できません。

同等の接続属性:

SQL_ATTR_KEEP_DYNAMIC

使用上の注意:

KeepDynamic CLI/ODBC 構成キーワードは、 DB2 for z/OS and OS/390 サーバー上での CLI パッケージのバインド方法に応じて設定する必要があります。
KeepDynamic は、次のように設定します。

- 0 - サーバー上の CLI パッケージが KEEPDYNAMIC NO オプションを指定してバインドされている場合
- 1 - サーバー上の CLI パッケージが KEEPDYNAMIC YES オプションを指定してバインドされている場合

KeepDynamic を使用する場合、CurrentPackageSet CLI/ODBC キーワードも設定することをお勧めします。それらのキーワードを併用する方法については、KEEPDYNAMIC サポートを有効にすることに関する資料を参照してください。

LOBCacheSize CLI/ODBC 構成キーワード

LOB の最大キャッシュ・サイズ (バイト単位) を指定します。

db2cli.ini キーワード構文:

LOBCacheSize = *正整数*

デフォルト設定:

LOB はキャッシュに入れられません。

同等の接続またはステートメント属性:

SQL_ATTR_LOB_CACHE_SIZE

使用上の注意:

このキーワードを設定すると、アンバインドされた LOB データを検索するときに LOB ロケーターを使用することを回避できます。例えば、アプリケーションが `SQLFetch()` を呼び出す前に列をバインドしないで、その後に `SQLGetData()` を呼び出して LOB を取り出す場合、LOBCacheSize が取り出される LOB 全体を含むことのできる十分な大きさに設定されていれば、LOB は LOB ロケーターではなく LOB キャッシュから検索されます。この場合、LOB ロケーターの代わりに LOB キャッシュを使用することによって、パフォーマンスが改善されます。

Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするサーバーは、LOB および XML データの戻りをデータの実際の長さに応じて最適化します。 LOB および XML データは、その全体を戻すか、またはプログレッシブ参照と呼ばれる内部トークンとして戻すことができます。 CLI はプログレッシブ参照のデータ検索を管理します。

Dynamic Data Format をサポートするサーバー上のデータを照会するアプリケーションでは、LOBCacheSize キーワードを設定すると、データの全体を戻すかまたはプログレッシブ参照として戻すかを決定するために使用されるしきい値が設定されます。データの長さが LOBCacheSize しきい値よりも大きい場合、プログレッシブ参照が管理のために CLI に戻されますが、データの長さが LOBCacheSize しきい値以下の場合は、データの全体が戻されます。

Dynamic Data Format をサポートしないサーバー上のデータを照会するアプリケーションの場合、LOBCacheSize しきい値は、CLI がメモリーのバッファーに入れる LOB の最大定義済みサイズを示します。 LOB の定義済みサイズが LOBCacheSize の設定値を超過する場合、LOB はキャッシュに入れられません。例えば、CLOB 列が 100MB で作成された表が現在 20MB のデータを保持していて、LOBCacheSize は 50MB に設定されているとします。この場合、LOB のサイズ (20MB) 自体は LOBCacheSize で設定されたサイズよりも小さいにもかかわらず、定義されている CLOB サイズ (100MB) が LOBCacheSize によって設定されてい最大キャッシュ・サイズ (50MB) より大きいため、 CLOB 列はキャッシュに入れられません。

ClientBuffersUnboundLOBS は関連キーワードです。

LOBFileThreshold CLI/ODBC 構成キーワード

SQLPutData() の使用時にバッファーに入れられる LOB データの最大バイト数を示します。

db2cli.ini キーワード構文:

LOBFileThreshold = *正整数*

デフォルト設定:

25 MB

使用上の注意:

このオプションは、SQLPutData() が呼び出されたときに、 CLI がメモリーのバッファーに入れる LOB データの最大バイト数を指定します。指定したキャッシュ・サイズが超過された場合、 LOB データがサーバーに送られる前に保持される一時ファイルがディスク上に作成されます。

LOBMaxColumnSize CLI/ODBC 構成キーワード

LOB データ・タイプの COLUMN_SIZE 列のデフォルト値をオーバーライドします。

db2cli.ini キーワード構文:

LOBMaxColumnSize = *ゼロよりも大きい整数*

デフォルト設定:

2 ギガバイト (DBCLOB については 1G)

次の場合にのみ適用可能:

LongDataCompat または LOB タイプの指定された MapXMLDescribe が使用されている。

使用上の注意:

SQL_CLOB、SQL_BLOB、と SQL_DBCLOB SQL、および SQL_XML SQL データ・タイプの COLUMN_SIZE 列について SQLGetTypeInfo() から戻される 2 ギガバイト (DBCLOB については 1G) の値をオーバーライドします。 SQL_XML では、MapXMLDescribe を LOB タイプに設定した LOBMaxColumnSize を指定する必要があります。 LOB 列を含む後続の CREATE TABLE ステートメントは、ここに設定する列サイズ値をデフォルトの代わりに使用します。

LoadXAInterceptor CLI/ODBC 構成キーワード

XA Interceptor をデバッグのためにロードします。

db2cli.ini キーワード構文:

LoadXAInterceptor = **0** | **1**

デフォルト設定:

XA Interceptor はロードされません。

使用上の注意:

このキーワードは、XA Interceptor をデバッグ目的で MTS にロードします。

LockTimeout CLI/ODBC 構成キーワード

LOCKTIMEOUT 構成パラメーターのデフォルト値を設定します。

db2cli.ini キーワード構文:

LockTimeout = **-1** | **0** | **正整数 ≤ 32767**

デフォルト設定:

タイムアウトはオフ (-1) であり、アプリケーションはロックが付与されるかデッドロックが発生するまでロックを待機します。

使用上の注意:

LockTimeout キーワードは、 CLI アプリケーションがロックの取得を待機する秒数を指定します。このキーワードが 0 に設定されている場合、ロックを待機することはありません。 -1 に設定されている場合、アプリケーションは、ロックが付与されるかデッドロックが発生するまで無期限に待機します。

LongDataCompat CLI/ODBC 構成キーワード

LOB を長いデータ・タイプまたはラージ・オブジェクト・タイプとして報告します。

db2cli.ini キーワード構文:

LongDataCompat = **0** | **1**

デフォルト設定:

LOB データ・タイプをラージ・オブジェクト・タイプとして参照します。

同等の接続属性:

SQL_ATTR_LONGDATA_COMPAT

使用上の注意:

このオプションは、アプリケーションがラージ・オブジェクト (LOB) 列を持つデータベースで作業を行うときに予期するデータ・タイプを CLI に示します。

このオプションの値は、次のとおりです。

- 0 = LOB データ・タイプをラージ・オブジェクト・タイプとして参照します。
- 1 = CLI/ODBC アプリケーションの場合にのみ、LOB を長いデータ・タイプとして報告します。

表 161. LOB データに対応するラージ・オブジェクトと長いデータ・タイプ

データベース・データ・タイプ	ラージ・オブジェクト (0 - デフォルト)	LONG データ・タイプ (1 — CLI/ODBC)
CLOB	SQL_CLOB	SQL_LONGVARCHAR
BLOB	SQL_BLOB	SQL_LONGVARBINARY

表 161. LOB データに対応するラージ・オブジェクトと長いデータ・タイプ (続き)

データベース・データ・タイプ	ラージ・オブジェクト (0 - デフォルト)	LONG データ・タイプ (1 — CLI/ODBC)
DBCLOB	SQL_DBCLOB	SQL_LONGVARGRAPHIC*

* LongDataCompat と共に MapGraphicDescribe キーワードが設定されている場合、 DBCLOB 列は、MapGraphicDescribe が 1 なら SQL_LONGVARCHAR、 MapGraphicDescribe が 2 なら SQL_WLONGVARCHAR の SQL タイプを戻します。

このオプションは、ラージ・オブジェクト・データ・タイプを扱うことができない ODBC アプリケーションを実行するときに役立ちます。

データに宣言されるデフォルト・サイズを小さくするために、 CLI/ODBC オプション LOBMaxColumnSize をこのオプションとともに使用することができます。

MapBigintCDefault CLI/ODBC 構成キーワード

BIGINT 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapBigintCDefault = **0** | **1** | **2**

デフォルト設定:

BIGINT データのデフォルトの C タイプ表記は SQL_C_BIGINT です。

使用上の注意:

MapBigintCDefault は、BIGINT 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、8 バイト整数を処理できない Microsoft アプリケーションで使用する必要があります。 MapBigintCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_BIGINT C タイプ表記の場合
- 1 - SQL_C_CHAR C タイプ表記の場合
- 2 - SQL_C_WCHAR C タイプ表記の場合

このキーワードは、 `SQLBindParameter()`、`SQLBindCol()`、`SQLGetData()` など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

MapCharToWChar CLI/ODBC 構成キーワード

SQL_CHAR、SQL_VARCHAR、および SQL_LONGVARCHAR に関連したデフォルトの SQL タイプを指定します。

db2cli.ini キーワード構文:

MapCharToWChar = **0** | **1**

デフォルト設定:

SQL_CHAR、SQL_VARCHAR、および SQL_LONGVARCHAR に対するデフォルトの SQL タイプ表現が使用されます。

同等の接続属性:

SQL_ATTR_MAPCHAR

使用上の注意:

MapCharToWChar は、SQL_CHAR、SQL_VARCHAR、および SQL_LONGVARCHAR 列またはパラメーター・マーカーについて記述するときに戻される SQL タイプを制御します。

MapCharToWChar は、次のように設定します。

- 0 - デフォルトの SQL タイプ表記を戻す場合
- 1 - SQL_CHAR を SQL_WCHAR として、SQL_VARCHAR を SQL_WVARCHAR として、および SQL_LONGVARCHAR を SQL_WLONGVARCHAR として戻す場合

MapCharToWChar の設定値は、次の CLI 関数にのみ影響します。

- SQLColumns()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()

MapDateCDefault CLI/ODBC 構成キーワード

DATE 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapDateCDefault = **0** | **1** | **2**

デフォルト設定:

DATE データのデフォルトの C タイプ表記は SQL_C_TYPE_DATE です。

使用上の注意:

MapDateCDefault は、 DATE 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 datetime 値のデフォルトの C タイプが SQL_C_CHAR であることを前提としている Microsoft アプリケーションで、使用する必要があります。 MapDateCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_TYPE_DATE C タイプ表記の場合: 年、月、日のそれぞれに対する数値メンバーを含む構造体。
- 1 - SQL_C_CHAR C タイプ表記の場合: "2004-01-01"
- 2 - SQL_C_WCHAR C タイプ表記の場合: "2004-01-01" (UTF-16)

このキーワードは、 SQLBindParameter()、SQLBindCol()、SQLGetData() など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

MapDateDescribe CLI/ODBC 構成キーワード

DATE 列および DATE 型のパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御します。

db2cli.ini キーワード構文:

MapDateDescribe = **0** | **1** | **2**

デフォルト設定:

DATE データのデフォルト SQL データ・タイプが戻されます (ODBC 2.0 の場合は SQL_DATE、 ODBC 3.0 の場合は SQL_TYPE_DATE)。

使用上の注意:

DATE 列および DATE 型のパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御するには、 MapDateDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合 (ODBC 2.0 の場合は SQL_DATE、 ODBC 3.0 の場合は SQL_TYPE_DATE)
- 1 - SQL_CHAR SQL データ・タイプを戻す場合
- 2 - SQL_WCHAR SQL データ・タイプを戻す場合

MapDateDescribe の設定値は、次の CLI 関数にのみ影響します。

- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

MapDecimalFloatDescribe CLI/ODBC 構成キーワード

DECFLOAT 列および DECFLOAT 型のパラメーター・マーカーの、デフォルトの C タイプおよび報告されるデータ・タイプを指定します。

db2cli.ini キーワード構文:

MapDecimalFloatDescribe = **0** | 1| 2| 3

デフォルト設定:

0

使用上の注意:

MapDecimalFloatDescribe は、データ・タイプが DECFLOAT の列およびパラメーターに使用される、デフォルトの C タイプを制御します。これは、C タイプの列またはパラメーターとして SQL_C_DEFAULT を指定できる CLI 関数の動作に影響します。そのような関数の例として、SQLBindParameter()、SQLBindCol()、および SQLGetData() があります。

MapDecimalFloatDescribe はまた、データ・タイプが DECFLOAT の列およびパラメーターに関して報告されるタイプを制御します。これは、パラメーターおよび列について情報を戻す CLI 関数に影響します。そのような関数の例として、SQLColAttribute() および SQLDescribeParam() があります。

この構成キーワードを使用するのは、10 進数浮動小数点タイプを処理できないアプリケーションに対して、または他のタイプよりも 10 進数浮動小数点タイプを処理することが多い場合です。

以下は、許可される値です。

表 162. MapDecimalFloatDescribe の有効値

値	DECFLOAT 列およびパラメーターが、このタイプであると報告される	DECFLOAT 列およびパラメーターが、このデフォルトの C タイプを使用する
0	SQL_DECFLOAT	SQL_C_CHAR
1	SQL_VARCHAR	SQL_C_CHAR
2	SQL_WVARCHAR	SQL_C_WCHAR
3	SQL_DOUBLE	SQL_C_DOUBLE

MapGraphicDescribe CLI/ODBC 構成キーワード

GRAPHIC、VARGRAPHIC、および LONGVARGRAPHIC 列およびこれらのデータ・タイプのパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを指定します。

db2cli.ini キーワード構文:

MapGraphicDescribe = **0** | **1** | **2**

デフォルト設定:

デフォルトの SQL データ・タイプが戻されます。つまり、GRAPHIC 列では SQL_GRAPHIC、VARGRAPHIC 列では SQL_VARGRAPHIC、 LONG VARGRAPHIC 列では SQL_LONGVARGRAPHIC です。

使用上の注意:

GRAPHIC ベースの列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御するには、 MapGraphicDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合
- 1 - CHAR ベースの SQL データ・タイプを戻す場合 (GRAPHIC 列では SQL_CHAR、 VARGRAPHIC 列では SQL_VARCHAR、および LONG VARGRAPHIC 列では SQL_LONGVARCHAR)
- 2 - WCHAR ベースの SQL データ・タイプを戻す場合 (GRAPHIC 列では SQL_WCHAR、 VARGRAPHIC 列では SQL_WVARCHAR、および LONG VARGRAPHIC 列では SQL_WLONGVARCHAR)

MapGraphicDescribe の設定値は、次の CLI 関数にのみ影響します。

- SQLDescribeCol()

- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

MapTimeCDefault CLI/ODBC 構成キーワード

TIME 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapTimeCDefault = **0** | **1** | **2**

デフォルト設定:

TIME データのデフォルトの C タイプ表記は SQL_C_TYPE_TIME です。

使用上の注意:

MapTimeCDefault は、 TIME 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 datetime 値のデフォルトの C タイプが SQL_C_CHAR であることを前提としている Microsoft アプリケーションで、使用する必要があります。 MapTimeCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_TYPE_TIME C タイプ表記の場合: 時、分、秒のそれぞれに対する数値メンバーを含む構造体。
- 1 - SQL_C_CHAR C タイプ表記の場合: "12:34:56"
- 2 - SQL_C_WCHAR C タイプ表記の場合: "12:34:56" (UTF-16)

このキーワードは、 SQLBindParameter()、SQLBindCol()、SQLGetData() など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

注: MapTimeCDefault は Patch2=24 を置き換えます。 MapTimeCDefault と Patch2=24 の両方が設定されている場合には、 MapTimeCDefault の値が優先されます。

MapTimeDescribe CLI/ODBC 構成キーワード

TIME 列および TIME 型のパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御します。

db2cli.ini キーワード構文:

MapTimeDescribe = **0** | **1** | **2**

デフォルト設定:

TIME データのデフォルト SQL データ・タイプが戻されます (ODBC 2.0 の場合は SQL_TIME、 ODBC 3.0 の場合は SQL_TYPE_TIME)。

使用上の注意:

TIME 列および TIME 型のパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御するには、 MapTimeDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合 (ODBC 2.0 の場合は SQL_TIME、 ODBC 3.0 の場合は SQL_TYPE_TIME)
- 1 - SQL_CHAR SQL データ・タイプを戻す場合
- 2 - SQL_WCHAR SQL データ・タイプを戻す場合

MapTimeDescribe の設定値は、次の CLI 関数にのみ影響します。

- `SQLColumns()`
- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`
- `SQLProcedureColumns()`
- `SQLSpecialColumns()`

MapTimestampCDefault CLI/ODBC 構成キーワード

TIMESTAMP 列およびパラメーター・マーカーのデフォルトの C タイプを指定します。

db2cli.ini キーワード構文:

MapTimestampCDefault = **0** | **1** | **2**

デフォルト設定:

TIMESTAMP データのデフォルトの C タイプ表記は SQL_C_TYPE_TIMESTAMP です。

使用上の注意:

MapTimestampCDefault は、 TIMESTAMP 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 datetime 値のデフォルトの C タイプが SQL_C_CHAR であることを前提としている Microsoft アプリケーションで、使用する必要があります。 MapTimestampCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_TYPE_TIMESTAMP C タイプ表記の場合: 年、月、日、時、分、秒、および秒の小数部分のそれぞれに対する数値メンバーを含む構造体。
- 1 - SQL_C_CHAR C タイプ表記の場合: 2004-01-01 12:34:56.123...*n* (*n*=12)。
- 2 - SQL_C_WCHAR C タイプ表記の場合: UTF-16 で 2004-01-01 12:34:56.123456...*n* (*n*=12)。

このキーワードは、 `SQLBindParameter()`、`SQLBindCol()`、`SQLGetData()` など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

MapTimestampDescribe CLI/ODBC 構成キーワード

TIMESTAMP 列および TIMESTAMP 型のパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御します。

db2cli.ini キーワード構文:

MapTimestampDescribe = **0** | **1** | **2**

デフォルト設定:

TIMESTAMP データのデフォルト SQL データ・タイプが戻されます (ODBC 2.0 の場合は SQL_TIMESTAMP、 ODBC 3.0 の場合は SQL_TYPE_TIMESTAMP)。

使用上の注意:

TIMESTAMP 列および TIMESTAMP 型のパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御するには、 MapTimestampDescribe を次のように設定します。

- 0 - デフォルトの SQL データ・タイプを戻す場合 (ODBC 2.0 の場合は SQL_TIMESTAMP、 ODBC 3.0 の場合は SQL_TYPE_TIMESTAMP)
- 1 - SQL_CHAR SQL データ・タイプを戻す場合
- 2 - SQL_WCHAR SQL データ・タイプを戻す場合

MapTimeStampDescribe の設定値は、次の CLI 関数にのみ影響します。

- `SQLColumns()`
- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`
- `SQLProcedureColumns()`
- `SQLSpecialColumns()`

MapXMLCDefault CLI/ODBC 構成キーワード

XML 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用されるデフォルトの C タイプ表記を制御します。

db2cli.ini キーワード構文:

MapXMLCDefault = **0** | **1** | **2** | **3**

デフォルト設定:

XML データのデフォルトの C タイプ表記は SQL_C_BINARY です。

使用上の注意:

MapXMLCDefault は、 XML 列およびパラメーター・マーカーに対して SQL_C_DEFAULT が指定されている場合に使用される C タイプを制御します。このキーワードは、主として Microsoft Access など、 XML 値のデフォルトの C タ

イプが SQL_C_WCHAR であることを前提としている可能性のある Microsoft アプリケーションで、使用する必要があります。 MapXMLCDefault は、次のように設定します。

- 0 - デフォルトの SQL_C_BINARY C タイプ表記の場合
- 1 - SQL_C_CHAR C タイプ表記の場合。XML データはローカル・アプリケーションのコード・ページに変換されるので、これによりデータが失われることがあります。
- 2 - SQL_C_WCHAR C タイプ表記の場合

このキーワードは、 SQLBindParameter()、SQLBindCol()、SQLGetData() など、SQL_C_DEFAULT を C タイプとして指定する CLI 関数の動作に影響を与えます。

MapXMLDescribe CLI/ODBC 構成キーワード

XML 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御します。

db2cli.ini キーワード構文:

MapXMLDescribe = **-370** | **-350** | **-152** | **-99** | **-98**

デフォルト設定:

XML データ用のデフォルト SQL データ・タイプが戻されます (SQL_XML (-370))。

使用上の注意:

XML 列およびパラメーター・マーカーが記述されている場合に戻される SQL データ・タイプを制御するには、MapXMLDescribe を次のいずれかの整数値に設定します。

- -370 は、デフォルトの SQL_XML SQL データ・タイプを戻す
- -350 は、SQL_DBCLOB SQL データ・タイプを戻す
- -152 は、SQL_SS_XML SQL データ・タイプを戻す

  注: SQL_SS_XML 値の -152 は、 Microsoft SQL Server の予約済み範囲に属していて、 IBM によって定義されていません。
- -99 は、SQL_BLOB SQL データ・タイプを戻す
- -98 は、SQL_CLOB SQL データ・タイプを戻す

LOB タイプにマップされた XML 値のデータ長は、マップされたデータ・タイプの最大長です。

値 1 に設定された LongDataCompat キーワードと共に使用するとき、 LOB データ・タイプにマップされた XML 値も対応する LONG データ・タイプにマップされます。

MapXMLDescribe に文字タイプが指定されていると、アプリケーションのコード・ページがソース・データ内のすべての文字をサポートしていない場合、データ変換

の際にデータが失われることがあります。そのため、XML 値を文字タイプにマップすることは、必ず注意して行うように推奨されています。

このキーワードは、XML 列に CLOB または BLOB としてアクセスするアプリケーションとの互換性のため、または Microsoft アプリケーション開発テクノロジーを使用するために推奨されます。

MaxLOBBlockSize CLI/ODBC 構成キーワード

LOB または XML データの戻りブロックの最大サイズを指定します。

db2cli.ini キーワード構文:

MaxLOBBlockSize = **0** | **...** | **2147483647**

デフォルト設定:

LOB または XML データのデータ・ブロック・サイズには限度はありません。

同等の接続またはステートメント属性:

SQL_ATTR_MAX_LOB_BLOCK_SIZE

使用上の注意:

データ検索時にサーバーは、最大ブロック・サイズに到達している場合でも、現在行に関するすべての情報をクライアントへの応答に含めます。

MaxLOBBlockSize と db2set レジストリー変数 DB2_MAX_LOB_BLOCK_SIZE の両方が指定されている場合、MaxLOBBlockSize の値が使用されます。

Mode CLI/ODBC 構成キーワード

デフォルトの接続モードを設定します。

db2cli.ini キーワード構文:

Mode = **SHARE** | **EXCLUSIVE**

デフォルト設定:

SHARE

次の場合には適用不可:

ホストまたは IBM Power Systems サーバーに接続している。

使用上の注意:

CONNECT モードを SHARE または EXCLUSIVE のいずれかに設定します。モードがアプリケーションによって接続時に設定される場合、この値は無視されます。デフォルトは SHARE です。

NotifyLevel CLI/ODBC 構成キーワード

診断レベルを設定します。

db2cli.ini キーワード構文:

NotifyLevel = **0** | **1** | **2** | **3** | **4**

デフォルト設定:

3

使用上の注意:

これは、db2cli.ini ファイルの [COMMON] セクションでのみ設定できます。

これは、データベース・マネージャーのパラメーター NOTIFYLEVEL と同等です。

OleDbReportIsLongForLongTypes CLI/ODBC 構成キーワード

OLE DB が、LONG データ・タイプに対して、DBCOLUMNFLAGS_ISLONG のフラグを設定するようにします。

db2cli.ini キーワード構文:

OleDbReportIsLongForLongTypes = **0** | **1**

同等の接続属性:

SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB

デフォルト設定:

LONG タイプ (LONG VARCHAR、LONG VARCHAR FOR BIT DATA、LONG VARGRAPHIC、および LONG VARGRAPHIC FOR BIT DATA) は、DBCOLUMNFLAGS_ISLONG フラグを設定しません。そのため、WHERE 節でその列が使用されることがあります。

使用上の注意:

OLE DB クライアント・カーソル・エンジンおよび OLE DB .NET Data Provider CommandBuilder オブジェクトは、IBM DB2 OLE DB Provider によって提供される列情報に基づいて、UPDATE および DELETE ステートメントを生成します。生成されたステートメントの WHERE 節に LONG タイプが含まれる場合には、ステートメントが失敗します。なぜなら、等価演算子を使った検索で LONG タイプは使用できないからです。キーワード OleDbReportIsLongForLongTypes を 1 に設定すると、IBM DB2 OLE DB Provider が LONG タイプ (LONG VARCHAR、LONG VARCHAR FOR BIT DATA、LONG VARGRAPHIC、および LONG VARGRAPHIC FOR BIT DATA) を、DBCOLUMNFLAGS_ISLONG フラグを設定して報告します。これにより、WHERE 節で long 列が使用されるのを防ぎます。

OleDbReportIsLongForLongTypes キーワードは、以下のデータベース・サーバーによってサポートされます。

- DB2 for z/OS
  - バージョン 6 (PTF UQ93891 適用済み)
  - バージョン 7 (PTF UQ93889 適用済み)
  - バージョン 8 (PTF UQ93890 適用済み)
  - バージョン 8 より後のバージョン (PTF は不要)
- DB2 Database for Linux, UNIX, and Windows
  - バージョン 8.2 (バージョン 8.1、フィックスパック 7 と等価) 以降

OleDbReturnCharAsWChar CLI/ODBC 構成キーワード

IBM DB2 OLE DB Provider が CHAR、VARCHAR、LONG VARCHAR、および CLOB データを記述する方法を指定します。

db2cli.ini キーワード構文:

OleDbReturnCharAsWChar = **0** | **1**

デフォルト設定:

IBM DB2 OLE DB Provider は、CHAR、VARCHAR、LONG VARCHAR、および CLOB データを DBTYPE_WSTR と記述します。

使用上の注意:

IBM DB2 OLE DB Provider はデフォルトで、CHAR、VARCHAR、LONG VARCHAR、および CLOB データを DB2 バージョン 8.1.2 の DBTYPE_WSTR として記述します。 CLI/ODBC 構成キーワード OleDbReturnCharAsWChar を使用すると、このデフォルトを変更して、これらの文字データ・タイプを DBTYPE_STR として報告できるようになります。

可能な設定値は次のとおりです。

- 0 - CHAR、VARCHAR、LONG VARCHAR、および CLOB データは DBTYPE_STR として記述され、 ISequentialStream 内のデータのコード・ページはクライアントのローカル・コード・ページになります。
- 1 - CHAR、VARCHAR、LONG VARCHAR、および CLOB データは DBTYPE_WSTR として報告され、 ISequentialStream 内のデータのコード・ページは UCS-2 になります。

OleDbSQLColumnsSortByOrdinal CLI/ODBC 構成キーワード

OLE DB の IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) が、ORDINAL_POSITION 列によってソートされた行セットを戻すようにします。

db2cli.ini キーワード構文:

OleDbSQLColumnsSortByOrdinal = **0** | **1**

同等の接続属性:

SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB

デフォルト設定:

IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) は、列 TABLE_CATALOG、TABLE_SCHEMA、TABLE_NAME、COLUMN_NAME によってソートされた行セットを戻します。

使用上の注意:

Microsoft OLE DB 仕様は、IDBSchemaRowset::GetRowset (DBSCHEMA_COLUMNS) が列 TABLE_CATALOG、TABLE_SCHEMA、TABLE_NAME、COLUMN_NAME によってソートされた行セットを戻すことを要求します。 IBM DB2 OLE DB Provider はその仕様に準拠します。ただし、Microsoft ODBC Bridge provider (MSDASQL) を使用するアプリケーションは、一般に、行セットが ORDINAL_POSITION によってソートされるようにコード化されています。

OleDbSQLColumnsSortByOrdinal キーワードを 1 に設定すると、プロバイダーが ORDINAL_POSITION によってソートされた行セットを戻すようにします。

OleDbSQLColumnsSortByOrdinal キーワードは、以下のデータベース・サーバーによってサポートされます。

- DB2 for z/OS
  - バージョン 6 (PTF UQ93891 適用済み)
  - バージョン 7 (PTF UQ93889 適用済み)
  - バージョン 8 (PTF UQ93890 適用済み)
  - バージョン 8 より後のバージョン (PTF は不要)
- DB2 Database for Linux, UNIX, and Windows
  - バージョン 8.2 (バージョン 8.1、フィックスパック 7 と等価) 以降

OnlyUseBigPackages CLI/ODBC 構成キーワード

CLI アプリケーションがビッグ・パッケージのみ使用できるようにします。

db2cli.ini キーワード構文:

OnlyUseBigPackages= 0 | 1

デフォルト設定:

CLI アプリケーションは、64 のセクションがあるスモール・パッケージを使用し (デフォルトでは 3 つ)、これらのセクションが使い尽くされると、384 のセクションがあるビッグ・パッケージの使用を開始します。

使用上の注意:

アプリケーションが、スモール・パッケージ内の 64 のセクションではなく、ビッグ・パッケージ内の多数のセクション (384) にアクセスできるようにするには、このオプションを使用できます。

OnlyUseBigPackages=1 キーワードを設定すると、スモール CLI パッケージを使用できません。

OptimizeForNRows CLI/ODBC 構成キーワード

'OPTIMIZE FOR n ROWS' 節をすべての SELECT ステートメントに付加します。

db2cli.ini キーワード構文:

OptimizeForNRows = *整数*

デフォルト設定:

節は付加されません。

同等のステートメント属性:

SQL_ATTR_OPTIMIZE_FOR_NROWS

使用上の注意:

このオプションは、"OPTIMIZE FOR n ROWS" 節をすべての SELECT ステートメントに付加します。 n は 0 よりも大きな整数です。0 (デフォルト) に設定された場合、この節は付加されません。

PWD CLI/ODBC 構成キーワード

デフォルト・パスワードを定義します。

db2cli.ini キーワード構文:

PWD = *パスワード*

デフォルト設定:

なし

使用上の注意:

パスワードが接続時にアプリケーションによって提供されない場合は、このパスワード を使用します。

パスワードは、プレーン・テキストとして db2cli.ini ファイルに保管されるため、保護されません。

DB2 for z/OS サーバーにアクセスする際には、パスワード・フレーズをパスワードとして使用できます。パスワード・フレーズは、大/小文字混合、数字、およびブランクを含めた特殊文字からなる文字ストリングです。パスワード・フレーズは、ほとんどのハッキングの試みに十分耐えられるだけ長く、それでいて記憶しやすいので書き留められそうもない点が、パスワードよりセキュリティーが優れています。

PWDPlugin CLI/ODBC 構成キーワード

ファイル DSN 接続または DSN なし接続でのクライアント側の認証に使用されるユーザー ID パスワード・プラグイン・ライブラリーの名前を指定します。

db2cli.ini キーワード構文:

PWDPlugin = *プラグイン名*

デフォルト設定:

デフォルト値はヌルで、DB2 の提供するユーザー ID パスワード・プラグイン・ライブラリーが使用されます。

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、クライアント側の接続認証に使用されるユーザー ID パスワード・プラグイン・ライブラリーの名前を指定します。このプラグインは、SERVER または SERVER_ENCRYPT 認証を使ってクライアントが認証されるときに使用されます。

Patch1 CLI/ODBC 構成キーワード

認識されている CLI/ODBC アプリケーション問題について、予備手段を指定します。

db2cli.ini キーワード構文:

Patch1 = { **0** | **1** | **2** | **4** | **8** | **16** | **...** }

デフォルト設定:

予備手段を使用しません。

使用上の注意:

このキーワードは、 ODBC アプリケーションの認識されている問題について予備手段を指定するために使用されます。値は、予備手段なしを指定するか、1 つまたは複数の予備手段について指定できます。ここに指定されたパッチ値は、一緒に設定できる Patch2 値とともに使用されます。

「CLI/ODBC 設定」ノートブックを使用して、使用する 1 つ以上のパッチを選択できます。 `db2cli.ini` ファイル自体に値を設定して複数のパッチ値を使用する場合は、単純に値を足してキーワード値を作成します。例えば、パッチ 1、4、および 8 を使用する場合は、Patch1=13 と指定してください。

- 0 = 予備手段なし (デフォルト)

表 163. Patch1 CLI/ODBC 構成キーワード値

値	説明
1	SQL ステートメントをデータベース・サーバーに送信する前に `COUNT(exp)` を `COUNT(*)` に置換する。 「互換モード」の DB2 for z/OS バージョン 7 およびバージョン 8.1 は、構文 `COUNT(exp)` をサポートしていません。PATCH1 = 1 を指定すると、この構文を使用するアプリケーションは修正なしで実行できます。置換される照会の複雑度によっては、置換が期待した結果にならない場合もあります。 この設定は、DB2 Database for Linux, UNIX, and Windows、および DB2 for z/OS のバージョン 8.1 より後のバージョンでは無視されます。
4	タイム・スタンプの時刻および小数部がゼロの場合に、入力のタイム・スタンプ・データを日付データに変更します。例えば、{ts 'YYYY-MM-DD 00:00:00'} は {d 'YYYY-MM-DD'} に変更されます。この値は通常、Microsoft Access の古いバージョンで必要になります。
8	タイム・スタンプの日付部分が 1899-12-30 または 1900-01-01 の場合に、入力のタイム・スタンプ・データを時間データに変更します。例えば、{ts '1899-12-30 HH:MM:SS'} は {t 'HH:MM:SS'} に変更されます。この値は通常、Microsoft Access の古いバージョンで必要になります。

表 163. Patch1 CLI/ODBC 構成キーワード値 (続き)

値	説明
64	出力の GRAPHIC ストリングをヌル終了します。この値は通常、2 バイト (DBCS) 環境の Microsoft Access で必要となります。
128	いくつかの Microsoft アプリケーションに関連した MSysConf 表のデフォルトのパフォーマンス最適化動作を無効にします。 Microsoft Access などの Microsoft アプリケーションは、MSysConf と呼ばれる構成表を使用します。これらのアプリケーションはデータベースに正常に接続された後に、通常「SELECT Config, nValue FROM MSysConf」という照会を発行します。MSysConf 表はデフォルトでは DB2 データベース内に存在しないので、この照会は「SQL0204N "MSysConf" は定義されていない名前です。」というエラーを出して失敗します。 Microsoft アプリケーションはこのエラーに対処して処理を続行できますが、ネットワークを越えて DB2 サーバーに照会を発行すると、さまざまなリソースが使用されます。 パフォーマンスを向上させるために、CLI はこの照会が常に失敗すると想定して、アプリケーションがこの照会を実行しようとすることを検出した場合に、SQLSTATE が S0002 のエラー (Table not found) を自動的に戻します。そのため、この照会はサーバーには送られません。ただし、ユーザーがデータベース内に MSysConf 構成表を作成していて、アプリケーションがそれにアクセスすることを望んでいる場合には、この PATCH1 値を設定してパフォーマンス最適化を無効にし、照会が実行されるようにすることができます。
256	保守のためにのみ使用
512	保守のためにのみ使用
1024	実行された UPDATE または DELETE ステートメントがどの行にも影響を与えなかった場合、 SQL_NO_DATA_FOUND の代わりに SQL_SUCCESS_WITH_INFO を SQLExecute() 関数および SQLExecDirect() 関数から戻します。この値は、いくつかの Microsoft Visual Basic アプリケーションで必要となることがあります。
4096	自動コミット・モードでカーソルを閉じた後に、COMMIT が発行されることを防止します。

表 163. Patch1 CLI/ODBC 構成キーワード値 (続き)

値	説明
8192	ストアード・プロシージャーを呼び出した後に、追加の結果セットを戻します。この追加の結果セットには 1 つの行があり、ストアード・プロシージャーの出力値から構成されます。この PATCH1 値は、追加の結果セットを必要とするいくつかの Powerbuilder アプリケーションによって必要とされることがあります。
32768	ドライバーが Microsoft Query アプリケーションを DB2 for z/OS シノニムと共に機能するようにします。
65536	推奨されない
131072	推奨されない
262144	推奨されない
524288	推奨されない
1048576	保守のためにのみ使用
2097152	保守のためにのみ使用

Patch2 CLI/ODBC 構成キーワード

既知の CLI および ODBC アプリケーション問題について、予備手段を指定します。

db2cli.ini キーワード構文:

Patch2 = "*patch value 1*, *patch value 2*, *patch value 3*, ..."

デフォルト設定:

予備手段を使用しません

使用上の注意:

0、1、または複数の予備手段の値を指定できます。ここで指定するパッチ値は、他に設定できる可能性のある **Patch1** 値とともに使用できます。

複数のパッチを指定するときには、コンマで区切られたストリングとして値を指定します (値が加算されて合計が使用される **Patch1** オプションとは異なります)。

- 0 = 予備手段なし (デフォルト)

Patch2 値 3、4、および 8 を設定するには、次を指定してください。

```
Patch2="3, 4, 8"
```

表 164. Patch2 CLI/ODBC 構成キーワード値

値	説明
1	推奨されない
3	保守のためにのみ使用
4	推奨されない
5	推奨されない

表 164. Patch2 CLI/ODBC 構成キーワード値 (続き)

値	説明
6	CLI が、両方向スクロール・カーソルはサポートされていないことを示すメッセージを戻すようにします。この設定は、LOB を活用するいくつかのアプリケーション (Visual Basic など)、またはアプリケーションによって明示的に要求されている場合でも両方向スクロール・カーソルを使用する必要がないか使用することを望まないいくつかのアプリケーションで必要とされます。
7	すべての GRAPHIC 列データ・タイプを CHAR 列データ・タイプにマップします。GRAPHIC 列の精度も倍になります。例えば、GRAPHIC(20) は CHAR(40) として報告されます。
8	スキーマ呼び出しでのカタログ検索引数を無視します。
11	SQLGetInfo() 関数は、カタログ名が Visual Basic ストアード・プロシージャーでサポートされていることを報告します。
12	推奨されない
13	db2cli.ini 初期設定ファイル内のキーワードが、出力接続ストリングに追加されないようにします。
14	推奨されない
15	文字出力で、デフォルト・ロケールの小数点の代わりにピリオド区切り記号が使用されるようにします。
16	推奨されない
17	推奨されない
18	リテラルを繰り返し使用する非効率なアプリケーションについて、リテラルをパラメーター・マーカーに置き換えることを試行します。これが適用されるのは、リテラルだけを使用する VALUES 節のある INSERT SQL ステートメントだけです。最良のソリューションは、パラメーター・マーカーを使用するようにアプリケーションを適切にコーディングすることです。 この値は、バージョン 9.7 以降では使用できません。

表 164. Patch2 CLI/ODBC 構成キーワード値 (続き)

値	説明
19	外部結合の ON 節から括弧を除去します。外部結合は ODBC エスケープ・シーケンスで、サーバーは DB2 for MVS/ESA バージョン 5 です。DB2 for MVS/ESA バージョン 5 は、外部結合節の ON 節内で括弧が許可される ODBC 構文を現在サポートしていません。この **Patch2** 値を設定すると、外部結合のエスケープ・シーケンスを DB2 for MVS/ESA バージョン 5 に対して使用できるようになります。この値は、サーバーが DB2 for MVS/ESA バージョン 5 の場合にのみ設定してください。
20	サーバーが DB2 for MVS/ESA のとき、CLI が BETWEEN 述部を書き直すようにします。DB2 for MVS/ESA は、両方のオペランドにパラメーター・マーカーを使用する BETWEEN 述部を現在サポートしていません。この **Patch2** 値を設定すると、(expression ? BETWEEN ?) は (expression >= ? and expression <= ?) と書き直されます。.
21	推奨されない
22	SQLGetInfo() 関数が SQL_OUTER_JOINS=NO および SQL_OJ_CAPABILITIES=0 を報告するようにします。これにより、サポートされていない場合にアプリケーションが外部結合を使用することは防止されて、外部結合の照会が失敗しないようになります。
23	推奨されない
24	TIME データを SQL_CHAR データとして報告します。このパッチ値は、 Microsoft Access アプリケーションの予備手段として使用されます。
25	DECIMAL 列の CHAR 表記から後続ゼロを除去します。 Microsoft Access アプリケーションの予備手段として使用されます。
28	推奨されない
29	1 > x > -1 となる DECIMAl 値 x のストリング表記から先行ゼロを除去します。いくつかの MDAC バージョンがある ADO アプリケーションの予備手段として使用されます。
30	ストアード・プロシージャーのキャッシング最適化を無効にします。
31	推奨されない
32	推奨されない

表 164. Patch2 CLI/ODBC 構成キーワード値 (続き)

値	説明
33	CHAR に変換されるときに、ODBC バージョンではなく、タイム・スタンプ・データの ISO バージョンを戻します。
34	推奨されない
38	ステートメント・キャッシングをオフにします。
42	FOR UPDATE 節がキー・セット・カーソルと共に使用されないようにします。デフォルトでは、ほとんどのアプリケーションは、キー・セット・カーソルが更新可能であるかのように動作します。しかし、更新可能なカーソルが必要ではない場合、この **Patch2** 値はカーソルを読み取り専用に (ただしスクロール可能のままで、他のユーザーによる変更に依存するように) します。
50	COMMIT が発行されるときではなく SQLFetch() 関数が実行されるときに、LOB ロケーターを解放します。この **Patch2** 値は、アプリケーションが LOB 列を SQLBindCol() 関数 (または同等の記述子 API) とバインドしないで LOB データをフェッチするときに、内部的に使用されるロケーターを解放します。その場合も、アプリケーションに明示的に戻されるロケーターは、アプリケーションによって解放する必要があります。この **Patch2** 値は、アプリケーションが SQLCODE = -429 (ロケーターがない) を受け取る事態を避けるために使用できます。
56	DB2 Universal Database™ for z/OS and OS/390 バージョン 7 またはそれより前のバージョンのようにそれをサポートしないサーバーで、 Early Close Cursors に対するクライアント・サポートを可能にします。
57	出力標識ポインターを提供しないで、 NULL 出力パラメーター値を戻すストアード・プロシージャーの呼び出しを可能にします。これは通常、Borland Delphi 製品の古いバージョンで適用されます。
58	この **Patch2** 値を使用することにより、データベースに挿入された DateTime 値のために発生した切り捨てエラーを、切り捨て警告にダウングレードできます。

表 164. Patch2 CLI/ODBC 構成キーワード値 (続き)

値	説明
61	データが SQL_CHAR データ・タイプからクライアントに提供されていた場合、右にスペースが埋め込まれることがあります。このパッチ値は、右に埋め込まれた 1 バイトのスペースを取り除きますが、 2 バイト・スペースは取り除きません。この動作は、部分的に Neon Shadow Driver の動作を模倣したものです。
66	Windows 環境の小数点に影響する地域の設定を、アプリケーションで取得することを許可します。地域設定は、通常はデフォルトで無視されます。**Patch2**=15 の場合、あるいは **db2set** レジストリー変数 **DB2TERRITORY** または **DB2CODEPAGE** が設定されている場合は、このパッチ値は無視されます。サポートされる小数点は、ピリオドとコンマのみです。
78	ソース LOB 値が DB2 Universal Database for z/OS and OS/390 バージョン 7.1 以降の DBCLOB 列にある場合の、SQLGetPosition() 関数の動作を変更します。この **PATCH2** 値を設定すると SQLGetPosition() 関数が SYSIBM.SYSDUMMY1 ではなく SYSIBM.SYSDUMMYU を照会するようになります。
81	IBM Data Server Driver for ODBC and CLI は、CLI の仕様に基づき、式の列序数を返します。デフォルトでは、CLI は、列序数を返します。ODBC ドライバーは、 ODBC の仕様に基づき、式の列名として空ストリングを返します。 **Patch2** CLI /ODBC 構成キーワードを 81 に設定すると、IBM Data Server Driver for ODBC and CLI が、式の列名として空ストリングを返すようになります。
82	CLI が、SQL_ATTR_RESET_CONNECTION 値の代わりに、SQL_ATTR_REPLACE_QUOTED_LITERALS 値の意味を使用するよう強制します。

Port CLI/ODBC 構成キーワード

ファイル DSN で、または DSN なし接続で使用される、サーバー・システムのサービス名およびポート番号を指定します。

db2cli.ini キーワード構文:

Port = *サービス名* | *ポート番号*

デフォルト設定:
なし

次の場合にのみ適用可能:
プロトコルが TCPIP に設定されている。

使用上の注意:

このオプションは、 DB2 を実行しているサーバーへのこのクライアント・マシンからの TCP/IP 接続に必要な属性を指定するために、 Hostname オプションとともに使用します。これらの 2 つの値は、Protocol オプションが TCPIP に設定されているときにのみ使用されます。

サーバー・システムのサービス名またはポート番号のどちらかを指定します。サービス名は、使用時にクライアント・マシンでの検索に使用できることが必要です。

ProgramID CLI/ODBC 構成キーワード

アプリケーションと接続を関連付ける最大長 80 バイトのユーザー定義文字ストリングを設定します。この属性を設定すると、DB2 for z/OS バージョン 8 以降では、動的 SQL ステートメント・キャッシュに挿入されたステートメントにこの ID が関連付けられます。

db2cli.ini キーワード構文:
ProgramID = **<string>**

デフォルト設定:
なし

同等の接続属性:
SQL_ATTR_INFO_PROGRAMID

使用上の注意:
CLI アプリケーションをモニターする場合、**ProgramID** を使用して、動的 SQL ステートメント・キャッシュに挿入されたステートメントを識別することができます。 **ProgamID** を使用することにより、接続およびステートメントのレベルで ID (最大 80 バイトのストリング) を指定できます。 **ProgamID** が設定された場合、`SQLGetConnectAttr()` API と `SQLGetStatementAttr()` API はともに、指定された **ProgamID** 値を返します。

注: この属性は、DB2 UDB for z/OS バージョン 8 以降または IBM Informix® Dynamic Servers (IDS) にアクセスする CLI アプリケーションでのみサポートされます。

ProgramName CLI/ODBC 構成キーワード

モニター時にサーバーでアプリケーションを識別するために使用されるクライアント・アプリケーションのデフォルト名を、ユーザー定義の名前に設定します。

db2cli.ini キーワード構文:
ProgramName = **<ストリング>** | **PID**

デフォルト設定:

アプリケーションはクライアントによって識別されます。デフォルトで、実行可能ファイル名の最初の 20 バイトが使用されます。

同等の接続属性:

SQL_ATTR_INFO_PROGRAMNAME

使用上の注意:

CLI アプリケーションをモニターする場合、 DB2 が割り当てるデフォルトの識別子ではなく、アプリケーションをユーザー定義のストリングによって識別することが役立つ場合があります。 ProgramName を使用することによりユーザーは、20 バイト以下のストリング、またはストリング "PID" (引用符は含まない) のいずれかとして識別子を指定できます。

CLI アプリケーションに対して ProgramName が "PID" に設定されている場合、アプリケーションの名前は、接頭部 "CLI" の後、アプリケーションのプロセス ID、および現行のアクティブ接続ハンドルで構成されます。つまり、CLI<pid>:<connectionHandle#> という形式です。 "PID" の設定値は、同じデータベースとの数多くの接続を使用して複数のアプリケーションを実行するアプリケーション・サーバーをモニターする場合に便利です。

(他の種類のアプリケーションに対して ProgramName キーワードが "PID" に設定されている場合、接頭部 "CLI" はそれぞれのアプリケーションの種類に対応する値になります。すなわち、 JDBC アプリケーションの場合は "JDBC"、 OLE DB アプリケーションの場合は "OLEDB"、および .NET アプリケーションの場合は "ADONET"。)

PromoteLONGVARtoLOB CLI/ODBC 構成キーワード

LONGVARxxx データ・タイプを xLOB データ・タイプにプロモートします。

db2cli.ini キーワード構文:

PROMOTELONGVARTOLOB = 0 | 1

デフォルト設定:

0

使用上の注意:

このオプションは、LONGVARxxxx 値が 32K を超える可能性がある場合のみ使用してください。列の戻りタイプは、より大きいデータ・サイズを保持するために xLOB にプロモートされます。

Protocol CLI/ODBC 構成キーワード

ファイル DSN に対して、または接続頻度の低い DSN で使用される通信プロトコル。

db2cli.ini キーワード構文:

Protocol = **TCPIP | TCPIP6 | TCPIP4 | IPC | LOCAL**

デフォルト設定:
なし

使用上の注意:
これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

TCP/IP はファイル DSN を使用するときにサポートされる唯一のプロトコルです。オプションをストリング TCPIP (スラッシュなし) に設定してください。

このオプションを設定するときには、以下のオプションも設定しなければなりません。

- Database
- ServiceName
- Hostname

Protocol を **IPC** または **LOCAL** に設定することにより、IPC 接続を指定できます。

Protocol = **IPC** | **LOCAL** の場合、Instance キーワードも設定する必要があります。

QueryTimeoutInterval CLI/ODBC 構成キーワード

照会タイムアウトのチェック間の遅延 (秒単位)。

db2cli.ini キーワード構文:
QueryTimeoutInterval = **0** | **5** | **正の整数**

デフォルト設定:
5 秒

使用上の注意:

アプリケーションは SQLSetStmtAttr() 関数を使用することにより、**SQL_ATTR_QUERY_TIMEOUT** ステートメント属性を設定できます。この属性により、実行の取り消しが試行されてアプリケーションに戻るまでの、 SQL ステートメントまたは XQuery 式の実行完了待ちの秒数が示されます。

QueryTimeoutInterval 構成キーワードは、照会が完了したかどうかのチェックの間の CLI ドライバーの待ち時間を指示するために使用されます。

例えば、**SQL_ATTR_QUERY_TIMEOUT** を 25 秒 (25 秒待った後にタイムアウト) に設定して、 **QueryTimeoutInterval** を 10 秒 (照会を 10 秒ごとにチェック) に設定するとします。照会は 30 秒後 (25 秒の制限の後の最初のチェック) にならないとタイムアウトになりません。 CLI は、実行中の照会ごとの状況を定期的に照会するスレッドを開始して、照会タイムアウトをインプリメントします。
QueryTimeoutInterval 値は、有効期限切れの照会のチェック間に、照会タイムアウト・スレッドが待機する時間の長さを指定します。これは、実行中の照会に対する非同期操作であるために、 **SQL_ATTR_QUERY_TIMEOUT** + **QueryTimeoutInterval** 秒ま

でに、指定された照会がタイムアウトにならない可能性もあります。この例で、最良の場合のタイムアウトは 26 秒であり、悪い場合のタイムアウトは 35 秒です。

SQL_ATTR_QUERY_TIMEOUT が小さすぎる値に設定されていて、照会がタイムアウトになってはならない場合があるかもしれません。アプリケーションが変更できない場合 (サード・パーティーの ODBC アプリケーション)、 **QueryTimeoutInterval** は 0 に設定可能で、 CLI ドライバーは **SQL_ATTR_QUERY_TIMEOUT** 設定を無視するので、アプリケーションに戻るまでに SQL ステートメントが実行を完了するのを待機します。

QueryTimeoutInterval が 0 に設定されている場合、アプリケーションが **SQL_ATTR_QUERY_TIMEOUT** を設定しようとすると、 SQLSTATE 01S02 (オプション値が変更された) になります。

このオプションは、初期設定ファイルの **COMMON** セクションでのみ有効なので、DB2 データベースへのすべての接続に適用されます。

一方、**QueryTimeoutInterval** を **SQL_ATTR_QUERY_TIMEOUT** の設定よりも大きい値に設定し、指定したインターバルでタイムアウトが発生しないようにすることができます。例えば、アプリケーションが **SQL_ATTR_QUERY_TIMEOUT** 値を 15 秒に設定している場合でも、サーバーは、照会の実行に少なくとも 30 秒を必要とし、**QueryTimeoutInterval** は値 30 に設定可能で、この照会が 15 秒後にタイムアウトにならないようにします。

注:

- .NET Framework は db2cli.ini ファイルの設定をサポートしていません。
- この CLI キーワードは、CLI API 呼び出しを使用するストアード・プロシージャーまたはルーチンの内部で使用される場合は無視されます。
- **QueryTimeoutInterval** のために LOAD が中断されることもあります。この場合は、SQL0952N ではなく SQL3005N が返されます。

ReadCommonSectionOnNullConnect CLI/ODBC 構成キーワード

NULL connect が db2cli.ini 初期設定ファイルの [COMMON] セクションを処理できるようにします。

db2cli.ini キーワード構文:

ReadCommonSectionOnNullConnect = **0** | **1**

デフォルト設定:

NULL connect は db2cli.ini 初期設定ファイルを処理しません。

使用上の注意:

DB2 CLI および DB2 .NET ストアード・プロシージャーと共に使用する場合、1 を指定して、ストアード・プロシージャーが db2cli.ini ファイルの [COMMON] セクションを読み取り、そのセクションにリストされたキーワードを使用できるようにします。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 データベースへのすべての接続に適用されます。)

ReceiveTimeout CLI/ODBC 構成キーワード

既存の接続において、試行を終了し、通信タイムアウト・エラーを生成する前に、サーバーからの応答を待つ時間を秒単位で指定します。

db2cli.ini キーワード構文:

ReceiveTimeout = **0** | **1** |**2** | **...** | **32767**

デフォルト設定:

クライアントは既存の接続でサーバーからの応答を、無制限に待機します。

同等の接続属性:

SQL_ATTR_RECEIVE_TIMEOUT

使用上の注意:

このキーワードは、接続の確立中は効果がなく、また TCP/IP プロトコルに対してのみサポートされます。

Reopt CLI/ODBC 構成キーワード

特殊レジスター、グローバル変数、またはパラメーター・マーカーを持つ SQL ステートメントの、照会最適化または再最適化を有効にします。

db2cli.ini キーワード構文:

Reopt = **2** | **3** | **4**

デフォルト設定:

照会の実行時に照会最適化は行われません。コンパイラーによって選択されるデフォルトの推定値が、特殊レジスター、グローバル変数、またはパラメーター・マーカーに対して使用されます。

同等の接続またはステートメント属性:

SQL_ATTR_REOPT

使用上の注意:

コンパイラーによって選択されるデフォルトの推定値の代わりに、特殊レジスター、グローバル変数、またはパラメーター・マーカーに対して照会実行時に使用できる値を使用することによって、最適化が生じます。キーワードの有効値は、次のとおりです。

- 2 = SQL_REOPT_NONE。これはデフォルトです。照会の実行時に照会最適化は行われません。コンパイラーによって選択されるデフォルトの推定値が、特殊レジスター、グローバル変数、またはパラメーター・マーカーに対して使用されます。デフォルトの NULLID パッケージ・セットは、動的 SQL ステートメントの実行に使用されます。
- 3 = SQL_REOPT_ONCE。照会最適化は、照会実行時に一度、初めて照会を実行するときに生じます。 NULLIDR1 パッケージ・セットが使用されますが、これは REOPT ONCE バインド・オプションとバインドされています。
- 4 = SQL_REOPT_ALWAYS。照会最適化または再最適化は、照会の実行時に、毎回必ず生じます。 NULLIDRA パッケージ・セットが使用されますが、これは REOPT ALWAYS バインド・オプションとバインドされています。

NULLIDR1 および NULLIDRA は予約済みパッケージ・セット名で、使用時にはそれぞれ REOPT ONCE および REOPT ALWAYS が暗黙指定されます。これらのパッケージ・セットは、以下のコマンドで明示的に作成する必要があります。

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

Reopt および CurrentPackageSet の両方のキーワードが指定されている場合には、CurrentPackageSet が優先します。

ReportPublicPrivileges CLI/ODBC 構成キーワード

SQLColumnPrivileges() および SQLTablePrivileges() 結果で PUBLIC 特権について報告します。

db2cli.ini キーワード構文:

ReportPublicPrivileges = **0** | **1**

デフォルト設定:

PUBLIC 特権は報告されません。

使用上の注意:

このキーワードは、SQLColumnPrivileges() および SQLTablePrivileges() 結果で、 PUBLIC グループに割り当てられた特権を PUBLIC がユーザーであったかのように報告するかどうかを指定します。

ReportRetryErrorsAsWarnings CLI/ODBC 構成キーワード

CLI エラー・リカバリー中に警告として発覚したエラーを戻します。

db2cli.ini キーワード構文:

ReportRetryErrorsAsWarnings = **0** | **1**

次の場合にのみ適用可能:

RetryOnError キーワードが 1 に設定されている場合。

デフォルト設定:

CLI エラー・リカバリー中に発覚したエラーをアプリケーションに戻しません。

使用上の注意:

デフォルトでは、CLI 再試行ロジックは、致命的ではないエラーから正常にリカバリーできる場合に、 SQL_SUCCESS を戻してそのエラーをアプリケーションから隠します。このようにしてアプリケーション・バインディング・エラーを隠すことができるため、デバッグの場合は、ReportRetryErrorsAsWarnings を 1 に設定することができます。この設定はエラー・リカバリーをオンに維持しますが、警告として発覚したエラーを強制的に CLI がすべてアプリケーションに戻すようにします。

RetCatalogAsCurrServer CLI/ODBC 構成キーワード

カタログ関数はカタログ列に対して CURRENT SERVER 値を戻すか、NULL 値を戻すかを指定します。

db2cli.ini キーワード構文:

RetCatalogAsCurrServer= **0** | 1

デフォルト設定:

ターゲット DBMS がカタログ列に NULL を戻す場合は、CURRENT SERVER 値で置換しません。

使用上の注意:

ターゲット DBMS に対するカタログ関数が、カタログ列に NULL 値を戻す場合、RetCatalogAsCurrServer を 1 に設定すると、DBMS は代わりに CURRENT SERVER 値を戻します。

- 0 = カタログ関数はカタログ列に NULL 値を戻します (デフォルト)。
- 1 = カタログ関数はカタログ列に NULL 値の代わりに CURRENT SERVER 値を戻します。

例えば、カタログ関数 SQLTables() が、TABLE_CAT 列の値が NULL 値である結果セットを戻すとします。 RetCatalogAsCurrServer を 1 に設定すると、DBMS は TABLE_CAT 列に CURRENT SERVER 値を戻します。

RetOleDbConnStr CLI/ODBC 構成キーワード

Mode CLI/ODBC 構成キーワードが、数値またはストリング値のどちらを戻すかを指定します。

db2cli.ini キーワード構文:

RetOleDbConnStr = **0** | **1**

デフォルト設定:

Mode CLI/ODBC 構成キーワードの値が、ストリングとして戻されます。

使用上の注意:

Mode CLI/ODBC 構成キーワードは、CONNECT モードを SHARE または EXCLUSIVE のいずれかに設定します。 OLE DB は Mode の値が、ストリング表現ではなく数値表現であることを予期します。 RetOleDbConnStr は、ストリング値と数値のどちらを戻すかを切り替えます。

可能な設定値は次のとおりです。

- 0 - Mode キーワードの SQLDriverConnect() および SQLBrowseConnect() によって戻される値は、SHARE または EXCLUSIVE のいずれか
- 1 - Mode キーワードの SQLDriverConnect() および SQLBrowseConnect() によって戻される値は 3 (SHARE の場合) または 12 (EXCLUSIVE の場合) のいずれか

例えば、RetOleDbConnStr=1 と設定し、共有接続に対して以下の入力接続ストリングを指定して SQLDriverConnect() または SQLBrowseConnect() を呼び出すと、

```
DSN=SAMPLE;MODE=SHARE
```

出力接続ストリングは、以下の形式になります。

```
DSN=SAMPLE;UID=;PWD=;MODE=3
```

RetOleDbConnStr=1 と設定し、排他的接続に対して以下の入力接続ストリングを指定して SQLDriverConnect() または SQLBrowseConnect() を呼び出すと、

```
DSN=SAMPLE;UID=NEWTON;PWD=SECRET;MODE=EXCLUSIVE
```

出力接続ストリングは、以下の形式になります。

```
DSN=SAMPLE;UID=NEWTON;PWD=SECRET;MODE=12
```

OLE DB アプリケーションが、SQLDriverConnect() および SQLBrowseConnect() によって戻される Mode キーワードの値にストリングを使用すると、OLE DB コンポーネント・サービスからエラーを受け取ります。 OLE DB コンポーネント・サービスは、キーワード Mode が数値を持つことを予期するため、エラーを戻します。 RetOleDbConnStr を 1 に設定すれば、この動作は避けられます。その場合、Mode の値は数値になるからです。

RetryOnError CLI/ODBC 構成キーワード

CLI ドライバーのエラー・リカバリー動作をオン/オフにします。

db2cli.ini キーワード構文:

RetryOnError = **0** | **1**

デフォルト設定:

CLI ドライバーが致命的ではないエラーのエラー・リカバリーを試行することを許可します。

使用上の注意:

デフォルトでは、CLI は、アプリケーション・パラメーターの不正なバインドなどの致命的ではないエラーからのリカバリーを、失敗した SQL ステートメントに関する追加情報を検索してからそのステートメントを再び実行することによって行います。検索される追加情報には、データベース・カタログ表からの入力パラメーター情報が含まれます。 CLI がエラーから正常にリカバリーできる場合は、デフォルトでは、アプリケーションへのエラー報告はされません。 CLI/ODBC 構成キーワード ReportRetryErrorsAsWarnings を使用することにより、エラー・リカバリー警告がアプリケーションに戻されるようにするかどうかを設定できます。

重要: いったん CLI が正常にエラー・リカバリーを完了すると、アプリケーションが通常とは異なる動作をする場合がありますが、それは CLI が元の SQLBindParameter() 関数呼び出しで提供された情報ではなく、それに続けて実行した SQL ステートメントに対するリカバリー中に収集されたカタログ情報を使用するためです。このような動作が望ましくない場合は、 RetryOnError を 0 に設定し、強制的に CLI がリカバリーを試行しないようにします。しかし、ステートメント・パラメーターを正しくバインドするように、アプリケーションを変更する必要があります。

ReturnAliases CLI/ODBC 構成キーワード

CLI スキーマ API がメタデータ結果で別名を報告するかどうかを制御します。

db2cli.ini キーワード構文:

ReturnAliases = 0 | 1

デフォルト設定:

1: デフォルトでは、メタデータ・プロシージャーにおいて行を修飾する時に別名が考慮されます。

使用上の注意:

このキーワードは、メタデータ・プロシージャーにおいて行を修飾する時に別名 (またはシノニム) が考慮されるかどうかを指定します。別名を考慮しない場合、この照会で適格となる追加の表を判別するためのコストがかかる基本表との結合を行わないため、パフォーマンスをかなり向上できます。

- 0 : メタデータ・プロシージャーにおいて行を修飾する時に別名が考慮されません (パフォーマンスは良くなります)。
- 1 : メタデータ・プロシージャーにおいて行を修飾する時に別名が考慮されます。

このキーワードにより以下の CLI API が影響を受けます。

- SQLColumns()
- SQLColumnPrivileges()
- SQLTables()
- SQLTablePrivileges()
- SQLStatistics()
- SQLSpecialColumns()
- SQLForeignKeys()
- SQLPrimaryKeys()

ReturnSynonymSchema CLI/ODBC 構成キーワード

CLI スキーマ API が、スキーマ・プロシージャー結果セットの TABLE_SCHEM 列で DB2 for z/OS のシノニムのスキーマ名を報告するかどうかを制御します。

db2cli.ini キーワード構文:

ReturnSynonymSchema = 0 | 1

デフォルト設定:

1: デフォルトでは、シノニムの作成者が TABLE_SCHEM 列に返されます。

使用上の注意:

有効な設定:

- 0 : プロシージャー結果セットの TABLE_SCHEM 列は NULL になります。
- 1 : プロシージャー結果セットの TABLE_SCHEM 列には、シノニムの作成者が入ります。

スキーマで修飾された名前を使用して DB2 for z/OS サーバー上のシノニムにアクセスすることはできません。このため、シノニムに関していうならば、DB2 for z/OS サーバーに対して実行している場合には、CLI スキーマ API 結果セットの TABLE_SCHEM 列の意味は異なります。

CLI スキーマ API を DB2 Database for Linux, UNIX, and Windows サーバーに対して使用する場合には、この CLI キーワードは効果がありません。

このキーワードにより以下の CLI API が影響を受けます。

- SQLColumns()
- SQLColumnPrivileges()
- SQLTables()
- SQLTablePrivileges()
- SQLStatistics()
- SQLSpecialColumns()
- SQLForeignKeys()
- SQLPrimaryKeys()

このキーワードを使用するためには、次のプログラム一時修正 (PTF) を DB2 for z/OS データベース・サーバーに適用している必要があります。

表 165. DB2 for z/OS の **ReturnSynonymSchema** *のための PTF*

DB2 for z/OS	PTF または APAR 番号
バージョン 7	UK13643
バージョン 8	UK13644
バージョン 9	

SQLOverrideFileName CLI/ODBC 構成キーワード

特定の SQL ステートメントに関する CLI ステートメント属性の設定値がリストされている指定変更ファイルの場所を指定します。

db2cli.ini キーワード構文:

SQLOverrideFileName = **<絶対または相対パス名>**

デフォルト設定:

指定変更ファイルは使用されません。

使用上の注意:

SQLOverrideFileName キーワードは、CLI ドライバーの読む指定変更ファイルの場所を指定します。指定変更ファイルには、特定の SQL ステートメントに適用される CLI ステートメント属性の値が含まれています。例えば、db2cli.ini ファイル内で、パスと指定変更ファイル名を使用して **SQLOverrideFileName** キーワードを指定できます。

```
[MyDatabase]
SQLOverrideFileName=C:¥temp¥myfile.txt
```

c:¥temp¥ パス内に myfile.txt という名前の指定変更ファイルがあり、このファイルには特定の SQL ステートメントに適用する CLI ステートメント属性の値が含まれています。指定変更ファイルの先頭は COMMON セクション ([COMMON]) で、**Stmts** と呼ばれるキーワードのみ含まれます。このキーワードによりオーバーライドするステートメントの数が分かります。以下のサンプル指定変更ファイル (myfile.txt) には次の 2 つのステートメントがあります。

```
[Common]
Stmts=2

[1]
StmtIn=SELECT * FROM Employee
StmtAttr=SQL_ATTR_BLOCK_FOR_NROWS=50;SQL_ATTR_OPTIMIZE_FOR_NROWS=1;

[2]
StmtIn=SELECT * FROM Sales
StmtAttr=SQL_ATTR_MAX_ROWS=25;
```

指定変更ファイルの [COMMON] セクションの **Stmts** で指定されている数値は、その指定変更ファイルに含まれる SQL ステートメントの数と同じです。

SaveFile CLI/ODBC 構成キーワード

既存の正常な接続を確立するために使われたキーワードの属性値を保管するための、DSN ファイルのファイル名を指定します。

db2cli.ini キーワード構文:

このキーワードは db2cli.ini ファイル内では設定できません。

SQLDriverConnect 内の接続ストリングで、このキーワードの値を以下のように指定できます。

SaveFile = *ファイル名*

SchemaList CLI/ODBC 構成キーワード

表情報の照会に使用されるスキーマを制限します。

db2cli.ini キーワード構文:

SchemaList = " '*schema1*', '*schema2*',... '*schemaN*' "

デフォルト設定:

なし

使用上の注意:

SchemaList キーワードは、DBMS 内のすべての表をリストするアプリケーションに、より制限されたデフォルトを提供して、そのアプリケーションのパフォーマンスを向上させるために使用されます。

データベースに定義されている表の数が多い場合に、スキーマ・リストを指定して、アプリケーションが表情報を照会する時間を削減し、アプリケーションでリストされる表の数を削減することができます。各スキーマ名は、単一引用符で囲んでコンマで区切る必要があり、大文字小文字が区別されます。ストリング全体を二重引用符で囲む必要もあります。例:

```
SchemaList="'USER1','USER2','USER3'"
```

DB2 for z/OS の場合、このリストに CURRENT SQLID を含めることもできますが、単一引用符は使用しません。例:

```
SchemaList="'USER1',CURRENT SQLID,'USER3'"
```

ストリングの最大長は 256 文字です。

このキーワードを **DBName** および **TableType** と併用して、情報が戻される表の数をさらに制限することができます。

既存のスキーマ名のリストと共に、**USRLIBL* または **ALL* を指定することで、非修飾ストアード・プロシージャー呼び出しの解決、およびカタログ API 呼び出しによるライブラリー検索が可能になります。*ALL、CLI は、接続されたデータベースですべての既存のスキーマを検索します。*ALL は、CLI のデフォルトです。 DB2 for i サーバーで *USRLIBL を指定すると、CLI はサーバー・ジョブの現行ライブラリーで検索します。

IBM i Access ODBC ドライバーからマイグレーションして、DBQ または DefaultLibraries 接続ストリングのキーワードで *USRLIBL を指定した場合、次の例に示すように、*USRLIBL を **SchemaList** キーワードのスキーマ名のリストに追加します。

```
[DSNSAMP]
SCHEMALIST="*USRLIBL"
```

security CLI/ODBC 構成キーワード

ファイル DSN に対して、または DSN なしの接続で SSL サポートを使用するかどうかを指定します。

db2cli.ini キーワード構文:

security = SSL

デフォルト設定:

なし。

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、TCP/IP 通信で SSL サポートを使用するかどうかを指定します。これを使用できるのは、TCPIP、TCPIP4、TCPIP6 のプロトコルに限られます。指定しない場合、SSL サポートなしの標準の TCP/IP が使用されます。

ServerMsgMask CLI/ODBC 構成キーワード

CLI がいつエラー・メッセージをサーバーから要求する必要があるのかを指定します。

db2cli.ini キーワード構文:

ServerMsgMask = **0** | **1** | **-2**| **-1**

デフォルト設定:

CLI は最初にローカルのメッセージ・ファイルを調べて、メッセージが検索可能かどうかを確認します。マッチングする SQLCODE が見つからない場合、CLI は情報をサーバーに要求します。

同等の接続属性:

SQL_ATTR_SERVER_MSGTXT_MASK

使用上の注意:

このキーワードは、 475 ページの『UseServerMsgSP CLI/ODBC 構成キーワード』に関連して使用されます。キーワードは、以下のように設定できます。

- **0 (デフォルト) =** SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST。CLI は最初にローカルのメッセージ・ファイルを調べて、メッセージが検索可能かどうかを確認します。マッチングする SQLCODE が見つからない場合、CLI は情報をサーバーに要求します。
- 1 = SQL_ATTR_SERVER_MSGTXT_MASK_WARNINGS。CLI は常に、警告に関するサーバーからのメッセージ情報を要求しますが、エラー・メッセージはローカルのメッセージ・ファイルから取得します。
- -2 = SQL_ATTR_SERVER_MSGTXT_MASK_ERRORS。CLI は常に、エラーに関するサーバーからのメッセージ情報を要求しますが、警告メッセージはローカルのメッセージ・ファイルから取得します。
- -1 = SQL_ATTR_SERVER_MSGTXT_MASK_ALL。CLI はエラー・メッセージおよび警告メッセージの両方について、常にサーバーからのメッセージ情報を要求します。

ServiceName CLI/ODBC 構成キーワード

ファイル DSN 接続または DSN なしの接続で使用される、サーバー・システムのサービス名およびポート番号。

db2cli.ini キーワード構文:

ServiceName = *サービス名* | *ポート番号*

デフォルト設定:

なし

次の場合にのみ適用可能:

プロトコルが TCPIP に設定されている。

使用上の注意:

このオプションは、 DB2 を実行しているサーバーへのこのクライアント・マシンからの TCP/IP 接続に必要な属性を指定するために、 Hostname オプションとともに使用します。これらの 2 つの値は、PROTOCOL オプションが TCPIP に設定されているときにのみ使用されます。

サーバー・システムのサービス名またはポート番号のどちらかを指定します。サービス名は、使用時にクライアント・マシンでの検索に使用できることが必要です。

SkipTrace CLI/ODBC 構成キーワード

CLI トレース情報を DB2 トレースから除外します。

db2cli.ini キーワード構文:

SkipTrace = **0** | **1**

デフォルト設定:

トレース関数をスキップしません。

使用上の注意:

このキーワードにより、DB2 トレース関数が CLI アプリケーションをバイパスし、パフォーマンスが向上します。したがって、DB2 トレース機能 db2trc がオンになっていて、このキーワードが 1 に設定されると、CLI アプリケーションの実行の情報はトレースに含まれません。

トレース情報を必要としない UNIX プラットフォームの実稼働環境では、SkipTrace をオンにすることをお勧めします。ただし、テスト環境ではトレース出力が役に立つこともあるので、詳細な実行情報を得たい場合は、このキーワードをオフにする (またはデフォルト設定のままにする) ことができます。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

SQLCODEMAP CLI/ODBC 構成キーワード

SQLCODE マッピングを使用するか、オフにするかを指定します。

db2cli.ini キーワード構文:

SQLCODEMAP = <*MAP*> | <*NOMAP*>

デフォルト設定:

MAP

使用上の注意:

このキーワードは、db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

キーワードの値が MAP に設定されている場合、SQLCODE マッピングが使用されます。NOMAP オプションを指定すると、SQLCODE マッピングはオフになります。

SSLClientLabel CLI/ODBC 構成キーワード

証明書認証で使用するための特定の証明書にマップされる固有の SSL ラベルを指定します。

db2cli.ini キーワード構文:

SSLClientLabel = <*label*>

デフォルト設定:

なし。

使用上の注意:

DB2 バージョン 9.7 フィックスパック 6 以降では、**SSLClientLabel** キーワードは証明書認証でのみ使用できます。このキーワードは、db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

証明書ベースの認証を構成して認証情報を指定する場合、db2dsdriver.cfg 構成ファイルまたは db2cli.ini 構成ファイルのいずれかに、**SSLClientLabel** キーワードを指定する必要があります。

認証パラメーターを CERTIFICATE オプションに設定した一方、**SSLClientLabel** キーワードを db2cli.ini 構成ファイル、db2dsdriver.cfg 構成ファイル、または接続ストリングに指定していない場合、エラー CLI0221E が返されます。認証パラメーターを **CERTIFICATE** オプションに設定していない場合に、**SSLClientLabel** キーワードを db2cli.ini 構成ファイル、db2dsdriver.cfg 構成ファイル、または接続ストリングに指定すると、エラー CLI0222E が返されます。

SSLClientKeystash CLI/ODBC 構成キーワード

ファイル DSN に対して、または DSN なしの接続で使用される SSL stash ファイルを指定します。

db2cli.ini キーワード構文:

SSLClientKeystash = <*fully qualified stash file path*>

デフォルト設定:

なし。

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、鍵データベース・ファイルの暗号化パスワードを格納する stash ファイル (.sth) の絶対パスを指定します。 stash ファイルは、SSL ハンドシェーク時に鍵データベース・ファイルにアクセスするために使用されます。 SSL プロトコル (**security**=SSL) が指定されている場合、このパラメーターが定義されている必要があります。

SSLClientKeystash キーワードは、**SSLClientKeystoreDBPassword** キーワードと同時に使用することはできません。 SSL プロトコルを指定する場合 (security=SSL)、接続ストリング、CLI 構成ファイル、db2cli.ini、またはデータ・サーバー・ドライバー構成ファイル db2dsdriver.cfg に、**SSLClientKeystash** または **SSLClientKeystoreDBPassword** のいずれかを指定する必要があります。そうしなければ、接続失敗のエラーが返されます。

注:

- **ssl_client_keystash** キーワードも、前のバージョンとの互換性を提供するためにサポートされています。
- **SSLClientKeystash** キーワードは、DB2 バージョン 9.7 フィックスパック 6 からサポートされています。

SSLClientKeystoredb CLI/ODBC 構成キーワード

ファイル DSN に対して、または DSN なしの接続で使用される SSL 鍵データベース・ファイルを指定します。

db2cli.ini キーワード構文:

SSLClientKeystoredb = <*fully qualified key file path*>

デフォルト設定:

なし。

使用上の注意:

これは、特定のデータ・ソースに関する db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

このパラメーターは、鍵データベース・ファイル (.kdb) の絶対パスを指定します。鍵データベース・ファイルは、サーバー個人証明書からの署名者証明書を格納します。

- 自己署名サーバー個人証明書の場合、署名者証明書は個人証明書の公開鍵です。
- 認証局 (CA) による署名のあるサーバー個人証明書の場合、署名者証明書は個人証明書に署名した CA のルート CA 証明書です。

SSL プロトコル (**security**=SSL) が使用されている場合、このパラメーターが定義されている必要があります。認証が行われるためには、サーバーの個人証明書の署名者証明書もクライアントに存在していなければなりません。

注:

- **ssl_client_keystoredb** キーワードも、前のバージョンとの互換性を提供するためにサポートされています。
- **SSLClientKeystoredb** キーワードは、DB2 バージョン 9.7 フィックスパック 6 からサポートされています。

SSLClientKeystoreDBPassword CLI/ODBC 構成キーワード

認証パラメーターが CERTIFICATE に設定されている場合、SSL 接続のパスワードを指定します。

db2cli.ini キーワード構文:

SSLClientKeystoreDBPassword = <パスワード>

デフォルト設定:

なし。

使用上の注意:

DB2 バージョン 9.7 フィックスパック 6 以降では、**SSLClientKeystoreDBPassword** キーワードは証明書認証でのみ使用できます。このキーワードは、db2cli.ini ファイルの [Data Source] セクション内、または接続ストリング内で設定できます。

SSLClientKeystash 構成パラメーターと **SSLClientKeyStoreDBPassword** 構成パラメーターを同時に使用することはできません。db2cli.ini 構成ファイルまたは db2dsdriver.cfg 構成ファイルのいずれかに、

SSLClientKeystash 構成パラメーターと **SSLClientKeyStoreDBPassword** 構成パラメーターの両方を指定すると、エラー CLI0219E が返されます。

StaticCapFile CLI/ODBC 構成キーワード

キャプチャー・ファイル名を指定して、ファイルが保管されるパスを任意で指定します。

db2cli.ini キーワード構文:

StaticCapFile = < **完全ファイル名** >

デフォルト設定:

なし - キャプチャー・ファイルを指定してください。

次の場合にのみ適用可能:

StaticMode が Capture または Match に設定されている

使用上の注意:

このキーワードは、キャプチャー・ファイル名を指定して、ファイルが保管されるディレクトリーを任意で指定するために使用されます。

StaticLogFile CLI/ODBC 構成キーワード

静的プロファイル・ログ・ファイル名を指定して、ファイルが保管されるディレクトリーを任意で指定します。

db2cli.ini キーワード構文:

StaticLogFile = < **完全ファイル名** >

デフォルト設定:

静的プロファイル・ログは作成されません。ファイル名をパス名なしで指定する場合は、現行パスが使用されます。

次の場合にのみ適用可能:

StaticMode が Capture または Match に設定されている

使用上の注意:

このキーワードは、静的プロファイル・ログ・ファイル名を指定して、ファイルが保管されるディレクトリーを任意で指定するために使用されます。

StaticMode CLI/ODBC 構成キーワード

この DSN のために CLI/ODBC アプリケーションが SQL をキャプチャーするか静的 SQL パッケージを使用するかを選択します。

db2cli.ini キーワード構文:

StaticMode = **DISABLED** | **CAPTURE** | **MATCH**

デフォルト設定:

無効 - SQL ステートメントのキャプチャーは行われず、静的 SQL パッケージは使用されません。

使用上の注意:

このオプションを使用すると、 CLI/ODBC アプリケーションがこの DSN について発行した SQL が処理される方法を以下のように指定することができます。

- DISABLED = 静的モードは無効です。特定の処理を行いません。 CLI/ODBC ステートメントは、変更のない動的 SQL として実行されます。これはデフォルトです。
- CAPTURE = キャプチャー・モード。 CLI/ODBC ステートメントを動的 SQL として実行します。 SQL ステートメントが正常に終了した場合、SQL ステートメントは、後で DB2CAP コマンドによってバインドされるためにファイル (キャプチャー・ファイルとして知られる) にキャプチャーされます。
- MATCH = 一致モード。 CLI/ODBC ステートメントを、StaticPackage に指定されたキャプチャー・ファイルに一致ステートメントが見つかった場合に、静的 SQL ステートメントとして実行します。キャプチャー・ファイルは、最初に DB2CAP コマンドでバインドされる必要があります。

StaticPackage CLI/ODBC 構成キーワード

静的プロファイル・フィーチャーで使用されるパッケージを指定します。

db2cli.ini キーワード構文:

StaticPackage = *collection_id.package_name*

デフォルト設定:

なし - パッケージ名を指定してください。

次の場合にのみ適用可能:

STATICMODE が CAPTURE に設定されている

使用上の注意:

このキーワードは、アプリケーションが一致モードで動作する時に使用されるパッケージを指定するために使用されます。キャプチャー・ファイルを作成するには、最初にキャプチャー・モードを使用する必要があります。

示されたパッケージ名の最初の 7 文字のみが使用されます。 1 バイトの接尾部が、それぞれの分離レベルを表すために以下のように追加されます。

- 0 - 非コミット読み取り (UR)
- 1 - カーソル固定 (CS)
- 2 - 読み取り固定 (RS)
- 3 - 反復可能読み取り (RR)
- 4 - コミットなし (NC)

StmtConcentrator CLI/ODBC 構成キーワード

バージョン 9.7 フィックスパック 3 以降、DB2 はこのキーワードをサポートしています。これは、リテラル値を含む動的ステートメントがステートメント・キャッシュを使用するかどうかを指定するキーワードです。

db2cli.ini キーワード構文:

StmtConcentrator = **OFF** | **WITHLITERALS**

デフォルト設定:

サーバーでステートメント集中に対して指定されているデフォルトの動作。

同等の環境または接続属性:

SQL_ATTR_STMT_CONCENTRATOR

使用上の注意:

このオプションは、リテラル値を含む動的ステートメントがステートメント・キャッシュを使用するかどうかを指定します。

- OFF - ステートメント・コンセントレーターは無効です。
- WITHLITERALS - サーバーがサポートしている状況の場合、ステートメント・コンセントレーターが有効になり、リテラル値を含む動的ステートメントはステートメント・キャッシュを使用します。例えば、ステートメントにパラメーター・マーカー、名前付きパラメーター・マーカー、またはリテラルとパラメーター・マーカーおよび名前付きパラメーター・マーカーの組み合わせが含まれている場合、ステートメント・コンセントレーターは有効になりません。

DB2 for z/OS サーバーのバージョン 10 より前のサーバーでこの属性を使用すると、要求は無視されます。

StreamGetData CLI/ODBC 構成キーワード

SQLGetData() 関数のデータ出力ストリームを最適化します。

db2cli.ini キーワード構文:

StreamGetData = **0** | **1**

デフォルト設定:

CLI は、クライアント上のすべてのデータをバッファーに入れます。

同等の接続またはステートメント属性:

SQL_ATTR_STREAM_GETDATA

使用上の注意:

Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) がサーバーによってサポートされていない場合、 StreamGetData キーワードは無視されます。データをバッファーに入れる必要のないアプリケーションで Dynamic Data Format をサポートするサーバー上のデータを照会している場合は、1 を指定することによって、データをバッファーに入れる必要がないことを示してください。 CLI クライアントは、データ出力ストリームを最適化します。

StreamGetData が 1 に設定されている場合、出力バッファーに入れて戻すためにまだ使用できるバイト数を CLI が判別できない場合、切り捨て発生時に SQLGetData() は長さとして SQL_NO_TOTAL (-4) を戻します。それ以外の場合、SQLGetData() は、まだ使用できるバイト数を戻します。

StreamPutData CLI/ODBC 構成キーワード

1 つのステートメント・ハンドルで SQLPutData() 関数呼び出しを通して渡されるデータのパフォーマンスを向上させます。そのために、内部の接続レベル通信バッファーにデータを直接書き込みます。

db2cli.ini キーワード構文:

StreamPutData = **0** | **1**

デフォルト設定:

接続レベル・バッファーにデータを直接書き込むのではなく、デフォルトのステートメント・レベル・バッファーに書き込みます。

使用上の注意:

デフォルトでは CLI は、 SQLPutData() 関数呼び出しを介して渡されたデータを内部のステートメント・レベル・バッファーに書き込みます。その後の SQLParamData() 呼び出しによって、そのバッファーの内容が内部の接続レベル通信バッファーに書き込まれて、サーバーに送られます。特定の接続上のターゲット・データベースに所定のポイント・イン・タイムにデータを挿入するのに 1 つのステートメント・ハンドルだけを使用する場合、 StreamPutData=1 と設定すれば、パフォーマンスを向上することができます。この場合、CLI は挿入データを接続レベル・バッファーに直接書き込むことになります。ただし、特定の接続上のターゲット・データベースに対して複数のステートメントが並行してデータを挿入する場合に StreamPutData=1 と設定すると、パフォーマンスが低下する可能性があり、そのために不測のアプリケーション・エラーが生じることがあります。それは、共有の接続レベル通信バッファー内のステートメントは、シリアライゼーションしやすくなるからです。

SysSchema CLI/ODBC 構成キーワード

SYSIBM スキーマの代わりに検索する代替スキーマを設定します。

db2cli.ini キーワード構文:

SysSchema = **代替スキーマ**

デフォルト設定:

照会する DB2 for z/OS が SYSIBM のときに使用されるデフォルトの表修飾子名。

使用上の注意:

このオプションは、DB2 for z/OS からシステム・カタログ情報を取得するために CLI および ODBC カタログ関数の呼び出しが発行されたときに、SYSIBM スキーマの代わりに検索する代替スキーマまたは表修飾子を示します。

システム管理者はこの新しいスキーマ名を使用して、以下のようなシステム・カタログ表の行のサブセットから構成される、ビューのセットまたは表のコピーを定義できます。

- SYSIBM.SYSCOLAUTH
- SYSIBM.SYSCOLUMNS

- SYSIBM.SYSDATATYPES
- SYSIBM.SYSFOREIGNKEYS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSKEYS
- SYSIBM.SYSKEYCOLUSES
- SYSIBM.SYSPARMS
- SYSIBM.SYSRELS
- SYSIBM.SYSROUTINES
- SYSIBM.SYSTABAUTH
- SYSIBM.SYSTABCONST
- SYSIBM.SYSTABLES
- SYSIBM.SYSSYNONYMS

例えば、システム・カタログ表のビューのセットまたは表のコピーが ACME スキーマ内にある場合、SYSIBM.SYSTABLES のビュー (または表のコピー) は ACME.SYSTABLES となり、**SysSchema** は ACME に設定する必要があります。

すべての表名のシステム・カタログを自動的に照会するアプリケーションでは、定義および使用するシステム・カタログ表のビューを制限すると、アプリケーションによってリストされる表の数が減少します。これにより、表名のサブセットが戻されることになるので、アプリケーションが表情報を照会するために必要な時間を短縮できます。

システム表に定義されたものと同じ索引をコピーに定義して、システム・カタログ表のコピーを定義しそれを使用することにより、アプリケーションがデータベースを照会するために必要な時間を短縮できます。

SchemaList、**TableType**、および **DBName** キーワードを **SysSchema** キーワードと共に使用して、情報が戻される表の数をさらに制限することができます。

どのシステム・カタログ表を **SysSchema** で使用できるか、および **SysSchema** の機能について詳しくは、下記のサイトにアクセスして APAR PK05102 資料を参照してください。

IBM メインフレームのサポート

そのサイトで “PK05102” を検索してください。

TableType CLI/ODBC 構成キーワード

表情報の照会時に戻される TABLETYPES のデフォルト・リストを定義します。

db2cli.ini キーワード構文:

TableType = " *'TABLE'* | *,'ALIAS'* | *,'VIEW'* | , *'INOPERATIVE VIEW'* | , *'SYSTEM TABLE'* | *,'SYNONYM'* "

デフォルト設定:

TABLETYPES のデフォルト・リストは定義されません。

使用上の注意:

データベースに定義されている表の数が多い場合に、表タイプ・ストリングを指定して、アプリケーションが表情報を照会する時間を削減し、アプリケーションでリストされる表の数を削減することができます。

値の数の制限はありません。それぞれのタイプを大文字で指定し、単一引用符で囲み、コンマで区切ってください。ストリング全体が二重引用符で囲まれている必要もあります。例:

```
TableType="'TABLE','VIEW'"
```

このオプションを DBNAME および SCHEMALIST とともに使用して、情報が戻される表の数をさらに制限することができます。

TableType は、データベース内の表、ビュー、別名、およびシノニムを検索する CLI 関数にデフォルトを提供するために使用します。アプリケーションが関数呼び出しで表タイプを指定せず、このキーワードが使用されなかった場合は、すべての表タイプの情報が戻されます。アプリケーションが関数呼び出しで表タイプ の値を提供した場合は、引数値がこのキーワード値をオーバーライドします。

TableType に TABLE 以外の値が組み込まれている場合は、 DBName キーワード設定で特定の DB2 for z/OS データベースに情報を制限することはできません。

TargetPrincipal CLI/ODBC 構成キーワード

ターゲット・サーバーの DB2 インスタンス所有者の完全修飾された Kerberos プリンシパル名を指定します。

db2cli.ini キーワード構文:

TargetPrincipal = **name/instance@REALM**

デフォルト設定:

なし

使用上の注意:

Windows 2000、Windows XP、および Windows Server 2003 では、完全修飾された Kerberos プリンシパル名は、次のいずれかの形式の DB2 サーバーのサービスのログオン・アカウントです。userid@DOMAIN、userid@xxx.xxx.xxx.com、または domain¥userid。例えば、DB2 サーバーのサービスのアカウントが *LocalSystem* の場合、TargetPrincipal は HOST/*host_name*@DOMAIN です (*host_name* は完全修飾されたホスト名、*DOMAIN* は大文字の完全修飾されたドメイン名)。それ以外の場合、TargetPrincipal は userid@DOMAIN です (*userid* は DB2 サーバーのサービスのアカウントのユーザー ID、*DOMAIN* は大文字の完全修飾されたドメイン名)。

このキーワードを、db2cli.ini ファイルまたは db2dsdriver.cfg ファイルに追加できます。このキーワードは、db2cli.ini ファイルの [DATA SOURCE] または [COMMON] セクションのデータ・ソースを設定できます。

TempDir CLI/ODBC 構成キーワード

一時ファイルに使用されるディレクトリーを定義します。

db2cli.ini キーワード構文:

TempDir = < *絶対パス名* >

デフォルト設定:

TEMP または TMP 環境変数によって指定されているシステム一時ディレクトリーを使用します。

使用上の注意:

ラージ・オブジェクト (CLOBS、BLOBS、およびその他のラージ・オブジェクト) を操作するときは、データ変換が起きた場合、またはデータがサーバーに少しずつ送られる場合に、情報を保管するためにクライアント・マシンに一時ファイルが作成されることがよくあります。このオプションを使用すると、このような一時ファイルのロケーションを指定することができます。何も指定しなかった場合は、システム一時ディレクトリーが使用されます。

キーワードは db2cli.ini ファイルのデータ・ソース特有のセクションに置かれます。構文は次のとおりです。

- TempDir= F:¥DB2TEMP

指定されたパスはすでに存在している必要があり、アプリケーションを実行しているユーザーはそのパスへのファイルに書き込みできる権限を持っている必要があります。 CLI ドライバーが一時ファイルを作成しようとして、パス名が無効の場合、または指定のディレクトリーに一時ファイルを作成できない場合には、HY507 の SQLSTATE が戻されます。

TimestampTruncErrToWarning CLI/ODBC 構成キーワード

TIMESTAMP の小数秒のオーバーフローに対して戻り値を設定します。

db2cli.ini キーワード構文:

TimestampTruncErrToWarning = **0** | **1**

デフォルト設定:

TIMESTAMP の小数秒のオーバーフローの結果、エラー (SQLSTATE 22007) が発生します。

使用上の注意:

TimestampTruncErrToWarning は、TIMESTAMP の小数秒のオーバーフローの結果、エラー (SQLSTATE 22007) が表示されるか警告 (SQLSTATE 01S07) が表示されるかを制御します。

TimestampTruncErrToWarning は、次のように設定します。

- 0 - デフォルトのエラー (SQLSTATE 22007) を返します。
- 1 - 警告 (SQLSTATE 01S07) を返します。

Trace CLI/ODBC 構成キーワード

CLI/ODBC トレース機能をオンにします。

db2cli.ini キーワード構文:

Trace = **0** | **1** | **db2trc**

デフォルト設定:

トレース情報は取得されません。

同等の環境属性:

SQL_ATTR_TRACE

使用上の注意:

このオプションに値 1 を設定すると、CLI/ODBC トレース・レコードは、TraceFileName 構成パラメーターによって指示されたファイルに付加されるか、または TracePathName 構成パラメーターによって指示されたサブディレクトリー内のファイルに付加されます。Trace CLI/ODBC 構成キーワードは、TraceFileName または TracePathName のいずれも設定されていない場合、有効にはなりません。

例えば、CLI/ODBC トレース・ファイルを、ディスクに直接書き込まれるようにセットアップするには、次のようにします。

```
[COMMON]
Trace=1
TraceFileName=E:¥TRACES¥CLI¥MONDAY.CLI
TraceFlush=1
```

このオプションに値 db2trc を設定すると、DB2 インスタンスまたは DB2 Administration Server (DAS) のトレース機能が開始されます。このトレース機能は、 **db2trc** コマンドによって制御され、操作情報を記録し、その情報を読み取り可能な形式にフォーマット設定します。このトレース機能をオンにすると (デフォルトはオフ)、システムのパフォーマンスに影響を与える可能性があります。従って、このトレース機能は、DB2 技術サービス担当者からの指示があった場合にのみ使用してください。もしくは、必要なだけの情報が記録されたら、すぐにオフにしてください。

Trace CLI/ODBC 構成キーワードを db2trc に設定すると、自動的に **db2trc on** コマンドが -cli オプションで実行されます。トレースを停止するには、ユーザーがコマンド `db2trc off` を実行する必要があります。

このトレース機能によって記録されたトレース・レコードから CLI/ODBC トレース・ファイルを作成するには、内部トレース・バッファーからトレース・レコードをダンプしてフォーマット設定する必要があります。例えば、アプリケーションのトレース終了後に、CLI/ODBC トレースをダンプしてフォーマット設定するには、以下のステップが必要です。

1. `db2trc dump <dump_filename>`
2. `db2trc off`
3. `db2trc fmt -cli <dump_filename> <ODBC-CLI_trace_filename>`

Trace CLI/ODBC 構成キーワードを値 db2trc に設定すると、値 1 に設定した場合よりもパフォーマンスは向上します。また、ダンプされる CLI トレース・ファイル

は、Trace に値 1 を設定して生成される CLI トレース・ファイルよりも小さくなります。Trace CLI/ODBC 構成キーワード値 db2trc を使用すると、**db2trc** トレース・コマンドはデフォルトのメモリー・トレース・バッファー・サイズで開始されます。規模が大きい CLI アプリケーションの場合、デフォルトのトレース・バッファー・サイズでは、トレース・レコードが先頭から上書きされるのを防ぐのに不十分である可能性があります。このような制限を回避するには、コマンド **db2trc on** に **-f** パラメーターを指定して実行し、ファイルにトレースするようにします。例えば、clitrc.dmp という名前のファイルに CLI をトレースするコマンドは、以下のとおりです。

```
db2trc on -f clitrc.dmp -m *.*.CLITRC.*.*
```

ファイルへのトレースが終了した後に、トレース・レコードをダンプする必要はありませんが、最終的な CLI トレース・ファイルを得るには、やはりフォーマット設定が必要です。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 データベースへのすべての接続に適用されます。)

TraceAPIList CLI/ODBC 構成キーワード

CLI トレース機能を使用する際、どの API をトレースするかを指定します。**TraceAPIList** キーワードを設定しない場合、すべての API がトレースされます。

db2cli.ini ファイルに **TraceAPIList** キーワードを設定するには、次のコマンドを実行します。

```
db2 update cli cfg for section common using TRACEAPILIST API ID,API ID...
```

API ID には、CLI API の名前に対応する整数が入ります。 /sqllib/include/sqlcli1.h ファイルには、以下に示す例のような API 名と ID のマッピングがあります。

```
#define  SQL_API_SQLALLOCHANDLE        1001
#define  SQL_API_SQLFREEHANDLE         1006
#define  SQL_API_SQLCLOSECURSOR        1003
#define  SQL_API_SQLENDTRAN            1005
#define  SQL_API_SQLCOLATTRIBUTE       6
#define  SQL_API_SQLGETSTMTATTR        1014
#define  SQL_API_SQLGETCONNECTATTR     1007
```

/sqllib/samples/cli/db2conn.c サンプル・プログラムは、SQLConnect、SQLDriverConnect、および SQLBrowseConnect などのいくつかの CLI 接続 API の動作を示します。 **TraceAPIList** キーワードを設定しない場合、生成された CLI トレースは、以下の例のようになります。

```
[ Process: 4453, Thread: 47717036514016 ]
[ Date & Time:          06/26/2009 03:14:51.158736 ]
[ Product:              QDB2/LINUXX8664 DB2 v9.7.0.1 ]
[ Level Identifier:     08020107 ]
[ CLI Driver Version:   09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location:  /home/mayprasa/sqllib/cfg/db2cli.ini ]
[ CLI Driver Type:      IBM DB2 Application Runtime Client ]
```

```
SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
phOutput=&00007fffb229a990 )
    ---> Time elapsed - 0 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +2.830000E-004 seconds

SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3,
cbParam=0 )
    ---> Time elapsed - +4.300000E-005 seconds

SQLSetEnvAttr( )
    <--- SQL_SUCCESS   Time elapsed - +1.800000E-005 seconds

SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fffb229a8e4 )
    ---> Time elapsed - +3.600000E-005 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +4.480000E-004 seconds

SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="", cbUID=-3, szAuthStr="",
   cbAuthStr=-3 )
    ---> Time elapsed - +2.000000E-005 seconds
( DBMS NAME="DB2/LINUXX8664", Version="09.07.0001", Fixpack="0x28020107" )
( Application Codepage=819, Database  Codepage=1208,
Database XML Codepage=1208,
   Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=1200,
XML Send/Recv Codepage=1208 )

SQLConnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.242704E+000 seconds
( DSN=""SAMPLE"" )
( UID=" " )
( PWD="" )
```

SQLAllocHandle API のみをトレースするには、以下のコマンドを実行します。

```
db2 update cli cfg for section common using TRACEAPILIST 1001
```

1001 は、SQLAllocHandle API の ID です。生成された CLI トレースは、以下の例のようになります。

```
[ Process: 5977, Thread: 47628919556832 ]
[ Date & Time:          06/26/2009 03:17:25.066017 ]
[ Product:              QDB2/LINUXX8664 DB2 v9.7.0.1 ]
[ Level Identifier:     08020107 ]
[ CLI Driver Version:   09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location:  /home/mayprasa/sqllib/cfg/db2cli.ini ]
[ CLI Driver Type:      IBM DB2 Application Runtime Client ]



SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
phOutput=&00007fff3657bc70 )
    ---> Time elapsed - 0 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +2.720000E-004 seconds

SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fff3657bbc4 )
    ---> Time elapsed - +8.000000E-005 seconds
```

```
SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +4.250000E-004 seconds

SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fff3657bbc4 )
    ---> Time elapsed - +1.303675E+000 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +1.920000E-004 seconds

SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fff3657bbc0 )
    ---> Time elapsed - +1.154550E+000 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +1.320000E-004 seconds
```

使用上の注意:

TraceAPIList キーワードを使用するには、CLI TRACE を使用可能に設定する必要があります。

TraceAPIList! CLI/ODBC 構成キーワード

CLI トレース機能を使用する際、どの API をトレースしないかを指定します。 **TraceAPIList!** キーワードを設定しない場合、すべての API がトレースされます。

db2cli.ini ファイルに **TraceAPIList!** キーワードを設定するには、次のコマンドを実行します。

```
"db2 update cli cfg for section common using 'TRACEAPILIST!'API ID,API ID..."
```

API ID には、CLI API の名前に対応する整数が入ります。 /sqllib/include/sqlcli1.h ファイルには、以下に示す例のような API 名と ID のマッピングがあります。

```
#define  SQL_API_SQLALLOCHANDLE        1001
#define  SQL_API_SQLFREEHANDLE         1006
#define  SQL_API_SQLCLOSECURSOR        1003
#define  SQL_API_SQLENDTRAN            1005
#define  SQL_API_SQLCOLATTRIBUTE       6
#define  SQL_API_SQLGETSTMTATTR        1014
#define  SQL_API_SQLGETCONNECTATTR     1007
```

注: このコマンドを実行する場合、コマンド全体を二重引用符で囲み、**TRACEAPILIST!** キーワードを単一引用符で囲む必要があります。

/sqllib/samples/cli/db2conn.c サンプル・プログラムは、SQLConnect、SQLDriverConnect、および SQLBrowseConnect などのいくつかの CLI 接続 API の動作を示します。 **TraceAPIList!** キーワードを設定しない場合、生成された CLI トレースは、以下の例のようになります。

```
[ Process: 4453, Thread: 47717036514016 ]
[ Date & Time:          06/26/2009 03:14:51.158736 ]
[ Product:              QDB2/LINUXX8664 DB2 v9.7.0.1 ]
[ Level Identifier:     08020107 ]
[ CLI Driver Version:   09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location:  /home/mayprasa/sqllib/cfg/db2cli.ini ]
```

```
[ CLI Driver Type:      IBM DB2 Application Runtime Client ]



SQLAllocHandle( fHandleType=SQL_HANDLE_ENV, hInput=0:0,
phOutput=&00007fffb229a990 )
    ---> Time elapsed - 0 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +2.830000E-004 seconds

SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3, cbParam=0 )
    ---> Time elapsed - +4.300000E-005 seconds

SQLSetEnvAttr( )
    <--- SQL_SUCCESS   Time elapsed - +1.800000E-005 seconds

SQLAllocHandle( fHandleType=SQL_HANDLE_DBC, hInput=0:1,
phOutput=&00007fffb229a8e4 )
    ---> Time elapsed - +3.600000E-005 seconds

SQLAllocHandle( phOutput=0:1 )
    <--- SQL_SUCCESS   Time elapsed - +4.480000E-004 seconds

SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="", cbUID=-3,
szAuthStr="",
   cbAuthStr=-3 )
    ---> Time elapsed - +2.000000E-005 seconds
( DBMS NAME="DB2/LINUXX8664", Version="09.07.0001", Fixpack="0x28020107" )
( Application Codepage=819, Database  Codepage=1208, Database XML Codepage=1208,
   Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=1200,
XML Send/Recv Codepage=1208 )

SQLConnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.242704E+000 seconds
( DSN=""SAMPLE"" )
( UID=" " )
( PWD="" )
```

SQLAllocHandle API がトレースされないようするには、次のコマンドを実行します。

```
"db2 update cli cfg for section common using 'TRACEAPILIST!' 1001"
```

1001 は、**SQLAllocHandle** API の ID です。生成された CLI トレースは、以下の例のようになります。

```
[ Process: 5442, Thread: 47530732661472 ]
[ Date & Time:          06/26/2009 05:11:10.794067 ]
[ Product:              QDB2/LINUXX8664 DB2 v9.7.0.1 ]
[ Level Identifier:     08020107 ]
[ CLI Driver Version:   09.02.0000 ]
[ Informational Tokens: "DB2 v9.7.0.1","n090609","LINUXAMD6497","Fixpack 1" ]
[ Install Path:
/view/mayprasa_db2_v97fp1_linuxamd64_n090609_trc42/vbs/INST ]
[ db2cli.ini Location:  /home/mayprasa/sqllib/cfg/db2cli.ini ]
[ CLI Driver Type:      IBM DB2 Application Runtime Client ]



SQLSetEnvAttr( hEnv=0:1, fAttribute=SQL_ATTR_ODBC_VERSION, vParam=3, cbParam=0 )
    ---> Time elapsed - 0 seconds

SQLSetEnvAttr( )
    <--- SQL_SUCCESS   Time elapsed - +2.200000E-005 seconds
```

```
SQLConnect( hDbc=0:1, szDSN="sample", cbDSN=-3, szUID="", cbUID=-3, szAuthStr="",
   cbAuthStr=-3 )
    ---> Time elapsed - +4.850000E-004 seconds
( DBMS NAME="DB2/LINUXX8664", Version="09.07.0001", Fixpack="0x28020107" )
( Application Codepage=819, Database  Codepage=1208, Database XML Codepage=1208,
   Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=1200,
XML Send/Recv Codepage=1208 )

SQLConnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.156099E+000 seconds
( DSN=""SAMPLE"" )
( UID=" " )
( PWD="" )

SQLDisconnect( hDbc=0:1 )
    ---> Time elapsed - +4.400000E-005 seconds

SQLDisconnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.197430E-001 seconds
```

使用上の注意:

TraceAPIList! キーワードを使用するには、CLI TRACE を使用可能に設定する必要があります。

TraceComm CLI/ODBC 構成キーワード

それぞれのネットワーク要求に関する情報をトレース・ファイルに組み込むかどうかを指定します。

db2cli.ini キーワード構文:

TraceComm = **0** | **1**

デフォルト設定:

0 - ネットワーク要求情報はキャプチャーされません。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceComm が (1) に設定されていると、それぞれのネットワーク要求に関する下記の情報がトレース・ファイルに含められます。

- クライアントで完全に処理された CLI 関数と、サーバーとの通信に関係した CLI 関数
- サーバーとの各通信で送受信されたバイト数
- クライアントとサーバーの間でのデータの通信に費やされた時間

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceErrImmediate CLI/ODBC 構成キーワード

レコードの生成時に、診断レコードを CLI/ODBC トレースに書き込むかどうかを指定します。

db2cli.ini キーワード構文:

TraceErrImmediate = **0** | **1**

デフォルト設定:

診断レコードがトレース・ファイルに書き込まれるのは、SQLGetDiagField() または SQLGetDiagRec() が呼び出された場合だけです。あるいは、取り出されていない診断レコードのあるハンドルについては、 "Unretrieved Error Message" (取り出されていないエラー・メッセージ) がトレース・ファイルに書き込まれます。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceErrImmediate=1 と設定し、診断レコードが生成された時点でそれを CLI/ODBC トレース・ファイルに書き込むなら、アプリケーションの実行中にエラーが発生したタイミングを調べることができます。 SQLGetDiagField() と SQLGetDiagRec() を使用して診断情報を取り出さないアプリケーションの場合、これは特に便利です。というのは、ハンドルに対して診断レコードが生成された場合、そのハンドルに対して次の関数が呼び出される前にそれらが取り出されたりトレース・ファイルに書き込んだりされなければ、それらの診断レコードは失われてしまうからです。

TraceErrImmediate=0 (デフォルトの設定値) の場合、診断レコードがトレース・ファイルに書き込まれるのは、アプリケーションが SQLGetDiagField() または SQLGetDiagRec() を呼び出して診断情報を取り出す場合だけです。このキーワードが 0 に設定されている場合、アプリケーションが関数呼び出しによって診断情報を取り出さないのであれば、ハンドルに対して次に関数が呼び出された時点で診断レコードが存在する場合に、 "Unretrieved Error Message" の項目がトレース・ファイルに書き込まれます。

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceFileName CLI/ODBC 構成キーワード

すべての CLI/ODBC トレース情報の書き込み先のファイルを指定します。

db2cli.ini キーワード構文:

TraceFileName = < **完全修飾ファイル名** >

デフォルト設定:

なし

次の場合にのみ適用可能:
Trace オプションがオンになっている。

使用上の注意:

指定されたファイルが存在しない場合は、ファイルが作成されるか、または新しいトレース情報がファイルの終わりに付加されます。ただし、目的のパスのファイルが存在している必要があります。

指定されたファイル名が無効の場合、またはファイルの作成または書き込みが不可能な場合、トレースは行われず、エラー・メッセージも戻されません。

このオプションは、Trace オプションがオンになっているときにのみ使用します。このオプションは、CLI/ODBC 構成ユーティリティーで設定すると自動的に実行されます。

このオプションを設定した場合、TracePathName オプションは無視されます。

CLI トレースはデバッグ目的でのみ使用する必要があります。オンにしたまま長時間が経過すると、CLI/ODBC ドライバーの実行がスローダウンし、トレース情報が非常に大きくなる可能性があります。

TraceFileName キーワード・オプションは、マルチプロセスまたはマルチスレッド・アプリケーションでは使用しないでください。その理由は、すべてのスレッドまたはプロセスに関するトレース出力が同じログ・ファイルに書き込まれ、各スレッドまたはプロセスに関する出力を解読することが難しくなるためです。さらに、共有トレース・ファイルへのアクセスを制御するために使用されるセマフォーにより、マルチスレッド・アプリケーションの動作が変更される可能性があります。デフォルトの DB2 CLI トレース出力ログ・ファイル名はありません。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 データベースへのすべての接続に適用されます。)

TraceFlush CLI/ODBC 構成キーワード

n 個の CLI/ODBC トレース入力後にディスクへの書き込みを強制します。

db2cli.ini キーワード構文:
TraceFlush = **0** | **正の整数**

デフォルト設定:
入力ごとに書き込みを行いません。

次の場合にのみ適用可能:
CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceFlush は、トレース情報が CLI トレース・ファイルに書き込まれる頻度を指定します。デフォルトでは、TraceFlush は 0 に設定されており、各 DB2 CLI トレース・ファイルは、トレース対象アプリケーションまたはスレッドが正常終了するま

で開かれたままになっています。アプリケーションが異常終了すると、トレース・ログ・ファイルに書き込まれていない一部のトレース情報が失われる可能性があります。

このキーワードを正の整数に設定して、 CLI ドライバーが指定されたトレース入力回数の後に適切なトレース・ファイルをクローズおよび再オープンするように強制します。 TraceFlush キーワードの値が小さいほど、アプリケーションのパフォーマンスへの CLI トレースの影響は大きくなります。 TraceFlush=1 と設定すると、パフォーマンスへの影響が最大になりますが、アプリケーションが次のステートメントに移る前に各項目が確実にディスクに書き込まれます。

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceFlushOnError CLI/ODBC 構成キーワード

エラー発生時に、すべての CLI/ODBC トレース項目をディスクに書き込むかどうかを指定します。

db2cli.ini キーワード構文:

TraceFlushOnError = **0** | **1**

デフォルト設定:

エラー発生時に、ただちに CLI/ODBC トレース項目を書き込むことはしません。

次の場合にのみ適用可能:

CLI/ODBC Trace オプションがオンになっている。

使用上の注意:

TraceFlushOnError=1 と設定すると、 CLI ドライバーは、エラーが検出されるたびにトレース・ファイルをクローズしてから再オープンします。 TraceFlushOnError がデフォルト値 (0) のままである場合、トレース・ファイルがクローズされるのは、アプリケーションが正常に終了した時点、または TraceFlush キーワードによって指定された時間間隔に達した場合だけです。 TraceFlushOnError=0 の場合にアプリケーション・プロセスが異常終了すると、貴重なトレース情報が失われてしまう可能性があります。 TraceFlushOnError=1 と設定するとパフォーマンスに影響を与えることがありますが、エラーに関連するトレース項目は確実にディスクに書き込まれるようになります。

このオプションは、Trace CLI/ODBC オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceLocks CLI/ODBC 構成キーワード

ロック・タイムアウトのみを CLI/ODBC トレースにトレースします。

db2cli.ini キーワード構文:

TraceLocks = **0** | **1**

デフォルト設定:

トレース情報はロック・タイムアウトのみに限定されません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceLocks が 1 に設定されていると、ロック・タイムアウトはトレース・ファイルに記録されます。

このオプションは、CLI/ODBC TRACE オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TracePIDList CLI/ODBC 構成キーワード

CLI/ODBC トレースの取得対象となるプロセス ID を制限します。

db2cli.ini キーワード構文:

TracePIDList = **<値の指定なし>** | **<コンマで区切られた処理 ID のリスト>**

デフォルト設定:

CLI/ODBC トレースが実行されている場合、すべてのプロセス ID がトレースされます。

使用上の注意:

このキーワードは、多数のプロセスを作成するアプリケーションに使用します。そのようなアプリケーションの CLI/ODBC トレースのキャプチャーは、多数のトレース・ファイルを生成する可能性があります。このキーワードを使用することにより、アプリケーションの特定の疑わしいプロセスのトレースを収集できます。

このキーワードに値が指定されていない場合、すべての処理 ID がトレースされます。すべての処理 ID をトレースしない場合は、 CLI/ODBC トレースの実行時にトレースする処理 ID のリストをコンマ区切りで指定してください。

アプリケーションを初期化する前に、TraceRefreshInterval キーワードを何らかの値に設定しなければなりません。そうしなければ、TracePIDList キーワードは有効になりません。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 データベースへのすべての接続に適用されます。)

TracePIDList キーワードを使用するには、以下のようにします。

1. db2cli.ini ファイルで Trace CLI/ODBC キーワードがゼロに設定されているか、または指定されていないことを確認します。
2. 次のように、TraceRefreshInterval CLI/ODBC キーワードを、db2cli.ini ファイルの [COMMON] セクションに追加します。

   ```
   [COMMON]
   TraceRefreshInterval=<some positive integer>
   ```

3. アプリケーションを開始します。
4. **ps** (UNIX および Linux ベースのオペレーティング・システム上で) などのオペレーティング・システム・コマンドを使用して、CLI/ODBC トレースを収集する対象プロセスのプロセス ID を判別します。
5. 以下のキーワードを組み込むことによって、CLI/ODBC トレースをオンにし、識別されるプロセス ID を db2cli.ini ファイルの [COMMON] セクションに追加します。

   ```
   [COMMON]
   Trace=1
   TracePathName=<fully-qualified subdirectory name>
   TracePIDList=<comma-delimited list of process IDs>
   ```

指定されたプロセス ID の情報を含む CLI/ODBC トレースは、TracePathName キーワードによって指定されたディレクトリーにあります。余分の空ファイルもあるかもしれませんが、それらは無視してかまいません。

TracePIDTID CLI/ODBC 構成キーワード

各項目がトレースされるごとにプロセス ID およびスレッド ID をキャプチャーします。

db2cli.ini キーワード構文:

TracePIDTID = **0** | **1**

デフォルト設定:

トレース入力のプロセス ID およびスレッド ID はキャプチャーされません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TracePIDTID が 1 に設定されると、キャプチャーされた項目ごとにプロセス ID およびスレッド ID がトレース・ファイルに記録されます。 Trace キーワードが有効で、複数のアプリケーションが実行中の場合に効果があります。これは、Trace により、すべての実行アプリケーションのトレース情報が単一のファイルに書き込まれるためです。 TracePIDTID を有効にすることにより、処理とスレッドによって記録された情報を見分けることができます。

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TracePathName CLI/ODBC 構成キーワード

個々の CLI/ODBC トレース・ファイルの保管に使用されるサブディレクトリーを指定します。

db2cli.ini キーワード構文:

TracePathName = < **完全修飾サブディレクトリー名** >

デフォルト設定:

なし

次の場合にのみ適用可能:

Trace オプションがオンになっている。

次の場合には適用不可:

TraceFileName オプションがオンになっている。

使用上の注意:

同じ DLL または共有ライブラリーを使用する各スレッドまたは処理は、指定のディレクトリーに別々の CLI/ODBC トレース・ファイルを作成します。トレース・ファイルの名前は、アプリケーション・プロセス ID とスレッド・シーケンス番号を連結したものにより、自動的に付けられます。

指定されたサブディレクトリーが無効の場合、またはサブディレクトリーの書き込みが不可能な場合、トレースは行われず、エラー・メッセージも戻されません。

このオプションは、Trace オプションがオンになっているときにのみ使用します。このオプションは、CLI/ODBC 構成ユーティリティーで設定すると自動的に実行されます。

このオプションは、CLI/ODBC オプション TraceFileName が使用された場合には無視されます。

CLI トレースはデバッグ目的でのみ使用する必要があります。オンにしたまま長時間が経過すると、CLI/ODBC ドライバーの実行がスローダウンし、トレース情報が非常に大きくなる可能性があります。

TraceFileName と TracePathName の両方が指定されている場合は、 TraceFileName キーワードが優先され、TracePathName は無視されます。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceRefreshInterval CLI/ODBC 構成キーワード

Trace キーワードと **TracePIDList** キーワードが db2cli.ini ファイルの Common セクションから読み取られるインターバル (秒単位) を設定します。

db2cli.ini キーワード構文:

TraceRefreshInterval = 0 | 正整数

デフォルト設定:

Trace キーワードと **TracePIDList** キーワードは、アプリケーションの初期化時にのみ db2cli.ini から読み取られます。

使用上の注意:

アプリケーションの初期化前にこのキーワードを設定すると、*n* 秒以内に CLI/ODBC トレースを動的にオフにすることができます。

注: アプリケーションの実行中に **TraceRefreshInterval** を設定しても、効果はありません。このキーワードを有効にするには、アプリケーションの初期化前に設定されている必要があります。

このキーワードが設定されている場合、**Trace** キーワードと **TracePIDList** キーワードのみが db2cli.ini ファイルからリフレッシュされます。他の CLI または ODBC 構成キーワードは再読み取りされません。

TraceRefreshInterval がゼロ以外の正整数値に設定されている場合、db2cli.ini をモニターするスレッドが作成されます。この状況では、データベースに接続しているアプリケーションがマルチスレッド・セーフである必要があります。そうでない場合、アプリケーションが予期しない動作をすることがあります。

このキーワードは、初期設定ファイルの Common セクションに含まれるため、DB2 へのすべての接続に適用されます。

注: この CLI キーワードは、CLI API 呼び出しを使用するストアード・プロシージャーまたはルーチンの内部で使用される場合は無視されます。

TraceStmtOnly CLI/ODBC 構成キーワード

動的 SQL ステートメントのみを CLI/ODBC トレースにトレースします。

db2cli.ini キーワード構文:

TraceStmtOnly = **0** | **1**

デフォルト設定:

トレース情報は動的 SQL ステートメントのみに限定されません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceStmtOnly が 1 に設定されると、動的 SQL ステートメントのみがトレース・ファイルに記録されます。

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceTime CLI/ODBC 構成キーワード

経過時間カウンターをトレース・ファイルにキャプチャーします。

db2cli.ini キーワード構文:

TraceTime = **1** | **0**

デフォルト設定:

経過時間カウンターがトレース・ファイルに組み込まれます。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceTime が 1 に設定されると、経過時間カウンターがトレース・ファイルにキャプチャーされます。以下に例を示します。

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
  ---> Time elapsed - +6.785751E+000 seconds ( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <--- SQL_SUCCESS Time elapsed - +2.527400E-002 seconds
```

パフォーマンスを向上させたり、トレース・ファイルをより小さくしたりするためには、 TraceTime を 0 に設定して、これをオフにします。以下に例を示します。

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <--- SQL_SUCCESS
```

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

TraceTimestamp CLI/ODBC 構成キーワード

CLI/ODBC トレースにタイム・スタンプ情報がある場合、どのタイプのタイム・スタンプ情報が記録されるかを指定します。

db2cli.ini キーワード構文:

TraceTimestamp = **0** | **1** | **2** | **3**

デフォルト設定:

タイム・スタンプ情報はトレース・ファイルに書き込まれません。

次の場合にのみ適用可能:

Trace オプションがオンになっている。

使用上の注意:

TraceTimeStamp をデフォルトの 0 以外の値に設定すると、現在のタイム・スタンプまたは絶対実行時が、トレース情報の各行の先頭に (DB2 CLI トレース・ファイ

ルに書き込まれる時点で) 追加されます。以下の設定値は、トレース・ファイルにキャプチャーされるタイム・スタンプ情報のタイプを示しています。
- 0 = タイム・スタンプ情報なし
- 1 = プロセッサー時刻と ISO タイム・スタンプ (絶対実行時 (秒およびミリ秒) の後にタイム・スタンプ)
- 2 = プロセッサー時刻 (絶対実行時 (秒およびミリ秒))
- 3 = ISO タイム・スタンプ

このオプションは、CLI/ODBC Trace オプションがオンになっているときにのみ使用します。

(このオプションは初期設定ファイルの共通セクションに含まれるため、DB2 へのすべての接続に適用されます。)

Trusted_Connection CLI/ODBC 構成キーワード

現在認証されているユーザーとの接続を確立できるようにするかどうかを指定します。

構文: Trusted_Connection=Yes

注: このキーワードは、db2cli.ini ファイル内で設定した場合には無効になります。 SQLDriverConnect() への接続ストリング内に指定してください。

デフォルト設定:

CLI は、現行の認証済みユーザーではなく、 SQLDriverConnect() への接続ストリング内に指定されたユーザー ID およびパスワード情報を使用します。

使用上の注意:

データベースに接続する CLI アプリケーションは、通常は関数 SQLDriverConnect() を使用して接続します。この関数の入力引数の 1 つは *DriverCompletion* 値で、これはダイアログを開くときを決定します。*DriverCompletion* 引数の有効値は、次のとおりです。
- SQL_DRIVER_PROMPT: ダイアログは常に開始されます。
- SQL_DRIVER_COMPLETE: ダイアログは、接続ストリング内の情報が不足しているときだけ開始されます。
- SQL_DRIVER_COMPLETE_REQUIRED: ダイアログは、接続ストリング内の情報が不足しているときだけ開始されます。必須情報しか要求されません。ユーザーは、必要な情報だけを要求されます。
- SQL_DRIVER_NOPROMPT: ユーザーは、情報を要求されません。接続ストリングに含まれている情報を使用して、接続が試行されます。情報が足りない場合、SQL_ERROR が返されます。

注: *DriverCompletion* 引数について詳しくは、SQLDriverConnect() についての文書を参照してください。

Kerberos 環境のものなど、いくつかのアプリケーションでは、ユーザーが DB2 サーバーにユーザー ID またはパスワードを指定しないで接続できなければならないことがあります。アプリケーションが

SQL_DRIVER_NO_PROMPT オプションを SQLDriverConnect() 呼び出しで使用する場合、接続はユーザー認証なしで試行されます。その場合、このキーワードは不要になります。

サード・パーティー製のアプリケーションが関係する場合、そのアプリケーションによって使用されるプロンプト・レベルが SQL_DRIVER_NO_PROMPT 以外のものであれば、 CLI は欠落情報をユーザーに要求するためのダイアログを開きます。 **Trusted_Connection** を Yes に設定すると、それを SQLDriverConnect() の入力接続ストリングに提供することにより ("Trusted_Connection=Yes")、CLI は接続ストリングにあるユーザー ID またはパスワードのストリング (ブランク・ストリングを含む) を無視し、接続関数のプロンプト・レベルを無視します。CLI は現行の認証済みユーザーを使用して、データベースへの接続を試行します。接続の試行が失敗した場合、ユーザーにユーザー ID およびパスワードの入力を求めるプロンプトが出されます。

このキーワードは、SQLDriverConnect() の接続ストリング内でのみ使用されます。それを db2cli.ini ファイル内に設定しても、効果はありません。

TxnIsolation CLI/ODBC 構成キーワード

デフォルトの分離レベルを設定します。

db2cli.ini キーワード構文:

```
TxnIsolation = ReadUncommitted | ReadCommitted | RepeatableRead | Serializable | NoCommit | 1 | 2 | 4 | 8 | 32
```

デフォルト設定:

2 または ReadCommitted (カーソル固定)

次の場合にのみ適用可能:

デフォルトの分離レベルが使用されています。このキーワードは、アプリケーションが明確に分離レベルを設定している場合は無効になります。

同等のステートメント属性:

SQL_ATTR_TXN_ISOLATION

使用上の注意:

分離レベルを以下に設定します。

- 1 = SQL_TXN_READ_UNCOMMITTED - 読み取り非コミット (非コミット読み取り)
- 2 = SQL_TXN_READ_COMMITTED (デフォルト) - コミット読み取り (カーソル固定)
- 4 = SQL_TXN_REPEATABLE_READ - 反復可能読み取り (読み取り固定)
- 8 = SQL_TXN_SERIALIZABLE - シリアライズ可能 (反復可能読み取り)
- 32 = SQL_TXN_NOCOMMIT - (コミットなし、DB2 Universal Database for AS/400® 専用。この設定は自動コミットと同様です)

括弧の中の用語は、SQL92 分離レベルに相当する IBM の用語です。コミットなしは SQL92 分離レベルではなく、IBM DB2 for IBM i でのみサポートされることに注意してください。

表 166. サポートされている分離レベル

分離レベル	キーワード	SQL92	IBM の用語
1	SQL_TXN_READ_UNCOMMITTED	読み取り非コミット	非コミット読み取り (Uncommitted read)
2	SQL_TXN_READ_COMMITTED (デフォルト)	コミット読み取り	カーソル固定
4	SQL_TXN_REPEATABLE_READ	反復可能読み取り	読み取り固定
8	SQL_TXN_SERIALIZABLE	Serializable	反復可能読み取り
32	SQL_TXN_NOCOMMIT	SQL92 分離レベルなし	コミットなし

下記のテキスト値を使用して、db2cli.ini ファイル内で *TxnIsolation* キーワードを設定できます。

- ReadUncommitted
- ReadCommitted
- RepeatableRead
- Serializable
- NoCommit

リストにないテキスト値を使用した場合、値は無視され、*TxnIsolation* はデフォルト値に設定されます。

このキーワードは、デフォルトの分離レベルが使用されている場合にのみ適用できます。アプリケーションが接続またはステートメント・ハンドルの分離レベルを明示的に設定している場合、このキーワード設定は無視されます。

UID CLI/ODBC 構成キーワード

デフォルトのユーザー ID を定義します。

db2cli.ini キーワード構文:

UID = *userid*

デフォルト設定:

なし

使用上の注意:

指定された *userid* 値は、接続時にユーザー ID がアプリケーションによって提供されない場合に使用されます。

Underscore CLI/ODBC 構成キーワード

下線文字 (_) をワイルドカードとして処理するかどうかを指定します。

db2cli.ini キーワード構文:

Underscore = **0** | **1**

デフォルト設定:

下線文字は、任意の 1 文字または 0 文字と一致します。

使用上の注意:

このキーワードは、下線文字 (_) がワイルドカードとして認識されるか、それとも下線文字としてのみ認識されるかを指定します。可能な設定値は次のとおりです。

- 0 - 下線文字は下線文字としてのみ処理されます。
- 1 - 下線文字は任意の 1 文字または 0 文字と一致するワイルドカードとして処理されます。

名前に下線文字が含まれるデータベース・オブジェクトがある場合、 Underscore を 0 に設定すると、パフォーマンスが向上することがあります。

このキーワードが適用されるのは、引数として検索パターンを受け入れる下記のカタログ関数だけです。

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLProcedureColumns()`
- `SQLProcedures()`
- `SQLTablePrivileges()`
- `SQLTables()`

カタログ関数は特定の引数についてのみ検索パターンを受け入れる場合があることに注意してください。詳細については、特定の関数の資料を参照してください。

UseOldStpCall CLI/ODBC 構成キーワード

カタログ式プロシージャーが呼び出される方法をコントロールします。

db2cli.ini キーワード構文:

UseOldStpCall = **0** | **1**

デフォルト設定:

GRANT EXECUTE がプロシージャーに関して付与される必要のある新しい CALL 方式を使用して、そのプロシージャーを呼び出します。

使用上の注意:

DB2 Universal Database バージョン 8 より前のバージョンでは、プロシージャーの呼び出し側が、プロシージャーから呼び出されるパッケージに対する EXECUTE 特権を持っている必要がありました。今回から、呼び出し側はプロシージャーについての EXECUTE 特権を持っている必要があり、プロシージャーの定義者のみが、必要とされるどのパッケージに関しても EXECUTE 特権を持っている必要があります。

このキーワードは、プロシージャーを呼び出すのにどの方式を使用するかをコントロールします。 UseOldStpCall をオンに設定すると、プリコンパイラーが CALL ステートメントのプロシージャーの解決に失敗した場合、使用すべきでない sqleproc() API を使用してプロシージャーが呼び出されます。このキーワードをオフにすると、GRANT EXECUTE がプロシージャーで付与される必要のある、そのプロシージャーが呼び出されます。

UseServerMsgSP CLI/ODBC 構成キーワード

DB2 for z/OS サーバーへの接続時に、ストアード・プロシージャーが呼び出されてメッセージ・テキストが検索されるかどうかを指定します。

db2cli.ini キーワード構文:

UseServerMsgSP = **0** | **1**

デフォルト設定:

CLI はサーバーのストアード・プロシージャーを使用してメッセージを戻すのではなく、ローカルのメッセージ・ファイルを使用します。

同等の接続属性:

SQL_ATTR_SERVER_MSGTXT_SP

使用上の注意:

CLI は、SQL_ATTR_SERVER_MSGTXT_SP 接続属性によって指示されるストアード・プロシージャーを呼び出します。この属性が設定されていない場合は、CLI は SYSIBM.SQLCAMESSAGE ストアード・プロシージャーを呼び出します。この属性が DSNACCMG に設定されている場合は、CLI は、DB2 for z/OS バージョン 7 サーバーへの接続時には DSNACCMG を呼び出し、DB2 for z/OS バージョン 8 以降への接続時には SYSIBM.SQLCAMESSAGE を呼び出します。

このキーワードを使用するアプリケーションは 445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』を設定して、 CLI がいつこのプロシージャーを呼び出してメッセージ情報をサーバーから検索するのかを指定する必要もあります。 445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』が設定されていない場合、デフォルトで最初にローカルのメッセージ・ファイルを調べます。使用できるオプションについての詳細は、「445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』」を参照してください。

DSNACCMG は、DB2 for z/OS バージョン 9 では推奨されておらず、将来のリリースで除去される可能性があります。
SQL_ATTR_SERVER_MSGTXT_SP が DSNACCMG に設定されている場合は、この属性を別のストアード・プロシージャーに設定してメッセージ・テキストを取り出してください。あるいは、ローカル・メッセージ・ファイルを使用するか、ServerMsgTextSP 構成キーワードを使用してください。

ServerMsgTextSP CLI/ODBC 構成キーワード

DB2 for z/OS からメッセージ・テキストを検索するために使用するストアード・プロシージャーを指定します。

db2cli.ini キーワード構文:

ServerMsgTextSP = ストアード・プロシージャー名

デフォルト設定:

CLI はサーバーのストアード・プロシージャーを使用してメッセージを戻すのではなく、ローカルのメッセージ・ファイルを使用します。

同等の接続属性:

SQL_ATTR_SERVER_MSGTXT_SP

使用上の注意:

このキーワードを使用するアプリケーションは 445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』を設定して、 CLI がいつこのプロシージャーを呼び出してメッセージ情報をサーバーから検索するのかを指定する必要もあります。 445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』が設定されていない場合、デフォルトで最初にローカルのメッセージ・ファイルを調べます。使用できるオプションについての詳細は、「 445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』」を参照してください。

UseServerMsgSP では SQL_ATTR_SERVER_MSGTXT_SP 接続属性で指定されたプロシージャーの呼び出しのオン/オフを切り替えられるのに対し、**ServerMsgTextSP** にはプロシージャーを明示的に指定する必要があるという点が、**ServerMsgTextSP** と 475 ページの『UseServerMsgSP CLI/ODBC 構成キーワード』 との間の相違点です。

WarningList CLI/ODBC 構成キーワード

警告にグレードを下げるエラーを指定します。

db2cli.ini キーワード構文:

WarningList = " '**xxxxx**', '**yyyyy**', ..."

デフォルト設定:

SQLSTATE のグレードを下げません。

使用上の注意:

エラーとして戻される多くの SQLSTATE のグレードを警告に下げることができます。それぞれを大文字で指定し、単一引用符で囲み、コンマで区切ります。ストリング全体が二重引用符で囲まれている必要もあります。以下に例を示します。

```
WarningList=" '01S02', 'HY090' "
```

XMLDeclaration CLI/ODBC 構成キーワード

XML データがアプリケーション変数に暗黙的にシリアライズされるとき、XML 宣言の生成を制御します。

db2cli.ini キーワード構文:

XMLDeclaration = **non-negative integer < 7** | **7**

デフォルト設定:

XML バージョンおよびエンコード属性を含む BOM および XML 宣言は、暗黙的なシリアライゼーションの際に生成されます。

使用上の注意:

XMLDeclaration キーワードは、XML データがアプリケーション・バッファーに暗黙的にシリアライズされるとき、 XML 宣言のどの要素がアプリケーション・バッファーの前に付加されるかを制御します。この設定値は、XMLSERIALIZE 関数の結果に影響を与えません。

以下の値は、暗黙的なシリアライゼーションの際に生成されるコンポーネントを表します。必要な各コンポーネントの値を加算して、このキーワードを設定してください。

0 出力バッファーに追加される宣言またはバイト・オーダー・マーク (BOM) はありません。

1 ターゲット・エンコードが UTF-16 または UTF-32 の場合、該当するエンディアン (リトル・エンディアンまたはビッグ・エンディアン) のバイト・オーダー・マーク (BOM) が出力バッファーの前に付加されます。 (UTF-8 BOM は存在しますが、ターゲット・エンコードが UTF-8 の場合でも、データベース・サーバーはそれを生成しません。)

2 XML バージョンだけを含む最小の XML 宣言が生成されます。

4 ターゲット・エンコードを識別するエンコード属性が、生成された XML 宣言に追加されます。そのため、このキーワードの値を計算するときに設定値の 2 も含められるときだけ、この設定値は効果があります。

例えば、暗黙的なシリアライゼーションの際に BOM および最小の XML 宣言 (エンコード属性のない) が生成されるようにするには、 `XMLDeclaration = 3` を設定します。ここで、3 は 1 (BOM の生成を示す値) と 2 (最小の XML 宣言の生成を示す値) との合計です。

宣言または BOM が生成されないようにするには、XMLDeclaration を `XMLDeclaration = 0` と設定します。

第 4 章 CLI アプリケーションでの環境、接続、およびステートメントの属性

環境、接続、およびステートメントには、それぞれ定義済みの属性 (またはオプション) の集まりがあります。アプリケーションですべての属性を照会することができますが、デフォルト値を変更できるのは一部の属性だけです。アプリケーションで属性値を変更して、CLI の動作を変更することができます。

環境ハンドルには、その環境での CLI 関数の動作を制御する属性があります。アプリケーションでは、SQLSetEnvAttr() を呼び出して属性の値を指定したり、SQLGetEnvAttr() を呼び出して現行属性値を得ることができます。SQLSetEnvAttr() は、環境ハンドルに接続ハンドルを割り振る前にのみ呼び出すことができます。それぞれの環境属性の詳細は、CLI 環境属性のリストを参照してください。

接続ハンドルには、その接続での CLI 関数の動作を制御する属性があります。変更できる属性は、次の種類に分かれます。

- 接続ハンドルが割り振られたら、いつでも設定できるもの
- 実際の接続の確立が完了する前に限り設定できるもの
- 接続が確立されたら、いつでも設定できるもの
- 接続が確立された後で、未解決のトランザクションまたはオープン・カーソルがない場合に限り設定できるもの

アプリケーションでは、 SQLSetConnectAttr() を呼び出して接続属性の値を変更したり、 SQLGetConnectAttr() を呼び出して属性の現行値を得ることができます。ハンドルが割り振られた後であればいつでも設定できる接続属性の例としては、自動コミット・オプション SQL_ATTR_AUTOCOMMIT があります。それぞれの接続属性の詳細は、CLI 接続属性のリストを参照してください。

ステートメント・ハンドルには、そのステートメント・ハンドルを使って実行する CLI 関数の動作を制御する属性があります。変更できるステートメント属性は、次の種類に分かれます。

- 複数の属性を設定可能であるが、現行では 1 つの特定の値にのみ限定されるもの
- ステートメント・ハンドルが割り振られた後でいつでも設定できるもの
- オープン・カーソルがそのステートメント・ハンドルにない場合に限り設定できるもの

アプリケーションでは、SQLSetStmtAttr() を呼び出して設定可能なステートメント属性の値を指定したり、 SQLGetStmtAttr() を呼び出して属性の現行値を得ることができます。それぞれのステートメント属性の詳細は、CLI ステートメント属性のリストを参照してください。

SQLSetConnectAttr() 関数を、ステートメント属性を設定するために使用することはできません。この関数は、バージョン 5 より前の CLI でサポートされていました。

アプリケーションの多くは、デフォルトの属性設定値だけを使用します。しかし、これらのデフォルト値の一部が、アプリケーションの特定のユーザーには適さない状況もありえます。デフォルト値によっては、CLI/ODBC 構成キーワードを設定することで変更できるものもあります。 CLI では、エンド・ユーザーのために、いくつかの構成キーワードを設定する 2 つの方式が備えられています。 1 番目の方式は、SQLDriverConnect() および SQLBrowseConnect() 関数に接続ストリングを入力するときにキーワードと新しいデフォルト属性値を指定するものです。 2 番目の方式は、CLI/ODBC 構成キーワードを使用した CLI 初期設定ファイル内の新しいデフォルト属性値の仕様にかかわるものです。

CLI 初期設定ファイルを使用して、そのワークステーション上のすべての CLI アプリケーションについてデフォルト値を変更することができます。
SQLDriverConnect() 接続ストリング中にデフォルト属性値を指定する方法がアプリケーションで使用できない場合は、エンド・ユーザーにとってこの方法がデフォルト値を変更する唯一の方法になります。 SQLDriverConnect() に指定されるデフォルト属性値によって、特定の接続に関する CLI 初期設定ファイル内の値がオーバーライドされます。

デフォルト値変更のメカニズムは、エンド・ユーザーの調整用です。アプリケーション開発者は、適切な属性設定関数を使用する必要があります。アプリケーションが属性設定やオプション関数を、初期設定ファイルや接続ストリングの指定とは違う値を指定して呼び出すと、初期デフォルト値はオーバーライドされ、新しい値が有効になります。

図 1 は、基本的な接続シナリオに属性関数を追加する様子を示しています。

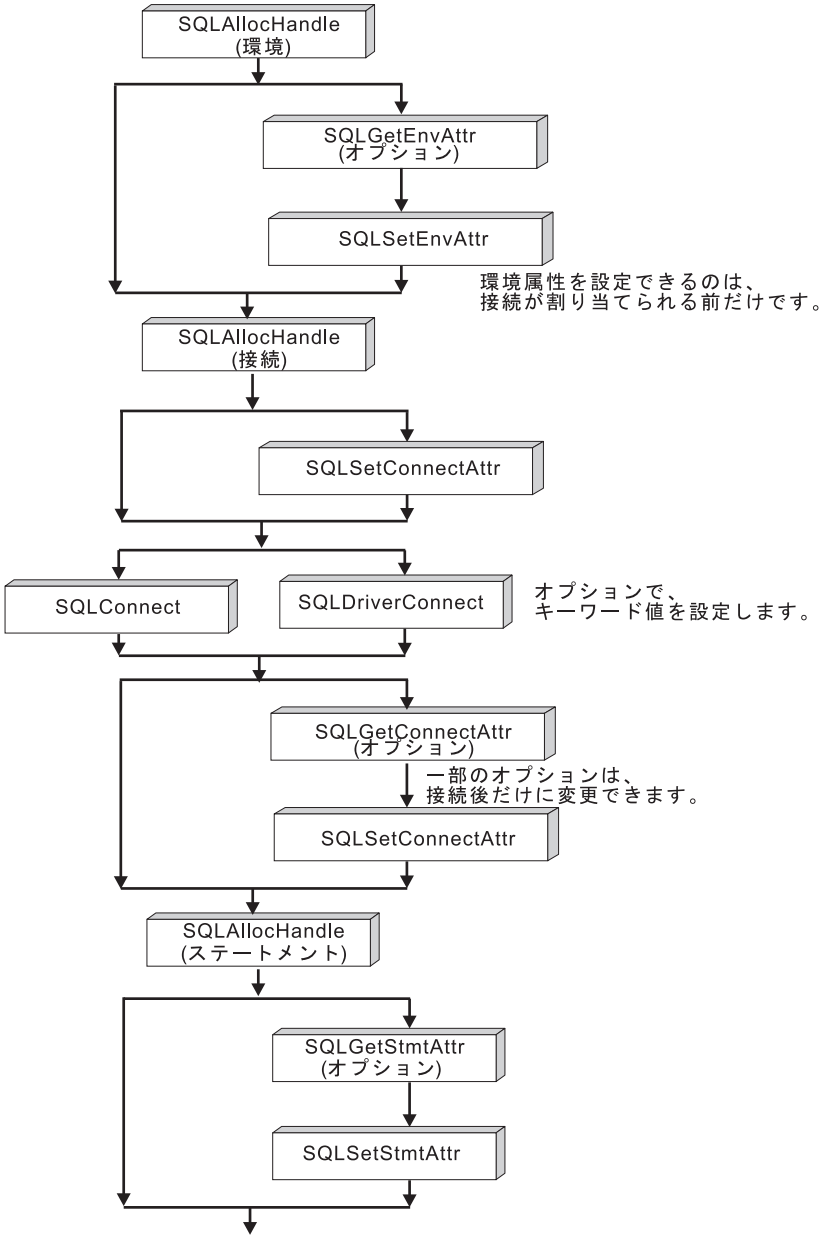


図 1. 属性 (オプション) の設定と取り出し

環境属性 (CLI) リスト

`SQLSetEnvAttr()` を使用して設定できる CLI 環境属性。

ODBC は、`SQLSetEnvAttr()` を使用したドライバー固有の環境属性の設定をサポートしていません。 CLI アプリケーションのみが、この関数を使って CLI 固有の環境属性を設定することができます。

SQL_ATTR_CONNECTION_POOLING

この属性は、DB2 UDB for Linux, UNIX, and Windows バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CONNECTTYPE

これは、SQL_CONNECTTYPE 属性に取って代わる属性です。このアプリケーションを整合分散環境で実行するか、それとも非整合分散環境で実行するかを指定する 32 ビット整数値。以下の値を指定することができます。

- **SQL_CONCURRENT_TRANS**: アプリケーションを使用して、1 つ以上のデータベースへの並行複数接続を行うことができます。各接続には、それぞれのコミット範囲があります。トランザクションの調整を行わせることはありません。あるアプリケーションが SQLEndTran() 上の環境ハンドルを使用してコミットを発行したが、すべての接続コミットが成功したわけではない場合、そのアプリケーションはリカバリーを行う必要があります。これはデフォルトです。
- **SQL_COORDINATED_TRANS**: アプリケーションは、複数のデータベース接続間でコミットとロールバックを調整できます。このオプション設定は、組み込み SQL のタイプ 2 CONNECT の指定に対応しています。前述の SQL_CONCURRENT_TRANS 設定とは対照的に、アプリケーションは 1 つのデータベースにつき 1 つのオープン接続のみを許可されます。

  注: この接続タイプでは、SQL_ATTR_AUTOCOMMIT 接続オプションのデフォルト値である SQL_AUTOCOMMIT_OFF の設定になります。

この属性をデフォルトから変更する場合、接続を環境ハンドルに対して確立する前にこれを設定する必要があります。

アプリケーションは通常、SQLSetEnvAttr() 関数を呼び出して、この属性を環境属性として設定します。 SQLSetEnvAttr() 関数は、環境ハンドルが割り当てられると同時に呼び出されます。ただし、ODBC アプリケーションは SQLSetEnvAttr() 関数にアクセスできないため、ODBC アプリケーションの場合には、個々の接続ハンドルが割り当てられてから接続が確立されるまでの間に、SQLSetConnectAttr() 関数を使用してこの属性を設定する必要があります。

環境ハンドル上のすべての接続の SQL_ATTR_CONNECTTYPE 設定は、同じでなければなりません。 1 つの環境で同時接続と整合接続の両方を使用することはできません。最初の接続のタイプが、それ以降のすべての接続のタイプを決定します。 SQLSetEnvAttr() は、接続アクティブに接続タイプを変更しようとすると、エラーが返されます。

384 ページの『ConnectType CLI/ODBC 構成キーワード』を使用して、デフォルト接続タイプを設定することもできます。

SQL_ATTR_CONNECTTYPE 属性は、IBM 定義の拡張機能です。

SQL_ATTR_CP_MATCH

この属性は、DB2 データベース・バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_DIAGLEVEL

説明 診断レベルを表す 32 ビット整数値。これは、データベース・マネージャーの DIAGLEVEL パラメーターと同等です。

値 有効な値は 0、1、2、3、または 4 (デフォルト値は 3)。

使用上の注意
この属性は、接続ハンドルを作成する前に設定しなければなりません。

SQL_ATTR_DIAGPATH

説明 診断データが格納されるディレクトリーの名前が入っているヌル終了文字ストリングを指すポインター。これは、データベース・マネージャーの DIAGPATH パラメーターと同等です。

値 デフォルト値は、UNIX および Linux オペレーティング・システムでは db2dump ディレクトリー、Windows オペレーティング・システムでは db2 ディレクトリーです。

使用上の注意
この属性は、接続ハンドルを作成する前に設定しなければなりません。

SQL_ATTR_INFO_ACCTSTR

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・アカウンティング・ストリングを識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS および DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 200 文字までの長さです。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_ACCTSTR 属性は、IBM 定義の拡張機能です。

SQL_ATTR_INFO_APPLNAME

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・アプリケーション名を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 32 文字までです。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_APPLNAME 属性は、IBM 定義の拡張機能です。

SQL_ATTR_INFO_USERID

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2

データベース製品の使用時に、データ・サーバーに送信されるクライアント・ユーザー ID を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 16 文字までです。このユーザー ID を認証ユーザー ID と混同しないでください。このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_USERID 属性は、IBM 定義の拡張機能です。

SQL_ATTR_INFO_WRKSTNNAME

説明 DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・ワークステーション名を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

値 値を設定すると、サーバーによっては、指定した長さ全体を処理することができず、値を切り捨てる場合があります。DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 サーバーがサポートするのは、最大 18 文字までです。ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

SQL_ATTR_INFO_WRKSTNNAME 属性は、IBM 定義の拡張機能です。

SQL_ATTR_MAXCONN

この属性は、DB2 バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_NOTIFYLEVEL

説明 通知レベルを表す 32 ビット整数値。これは、データベース・マネージャーの `NOTIFYLEVEL` パラメーターと同等です。

値 有効な値は 0、1、2、3、または 4 (デフォルト値は 3)。

使用上の注意

この属性値は、接続ハンドルを作成する前に設定しなければなりません。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_ODBC_VERSION

説明 特定の機能が ODBC 2.x (CLI v2) または ODBC 3.0 (CLI v5) の動作を示すかどうかを決定する 32 ビット整数。 ODBC アプリケ

ーションでは、SQLHENV 引数が指定されている関数を呼び出す前に、この環境属性を設定しないと、呼び出しは SQLSTATE HY010 (関数のシーケンス・エラーです。) を戻します。

値 次の値を使用して、この属性値を設定します。

- SQL_OV_ODBC3: この値により、次の ODBC 3.0 (CLI v5) 動作が発生します。
  - CLI は、日付、時刻、およびタイム・スタンプに、ODBC 3.0 (CLI v5) コードを戻し、これらのコードを予期しています。
  - SQLError()、SQLGetDiagField()、または SQLGetDiagRec() 関数が呼び出されると、CLI は ODBC 3.0 (CLI v5) SQLSTATE コードを戻します。
  - SQLTables() 関数への呼び出しの *CatalogName* 引数は検索パターンを受け入れます。
- SQL_OV_ODBC2: この値により、次の ODBC 2.x (CLI v2) 動作が発生します。
  - CLI は、日付、時刻、およびタイム・スタンプに ODBC 2.x (CLI v2) コードを戻し、これらのコードを予期しています。
  - SQLError()、SQLGetDiagField()、または SQLGetDiagRec() 関数が呼び出されると、CLI は ODBC 2.0 (CLI v2) SQLSTATE コードを戻します。
  - SQLTables() 関数への呼び出しの *CatalogName* 引数は検索パターンを受け入れません。
- SQL_OV_ODBC3_80: この値により、次の ODBC 3.0 (CLI v5) 動作が発生します。
  - CLI は、日付、時刻、およびタイム・スタンプに ODBC 3.x コードを戻し、これらのコードを予期しています。
  - SQLError()、SQLGetDiagField()、または SQLGetDiagRec() 関数が呼び出されると、CLI は ODBC 3.x SQLSTATE コードを戻します。
  - SQLTables() 関数への呼び出しの *CatalogName* 引数は検索パターンを受け入れます。

SQL_ATTR_OUTPUT_NTS

説明 出力引数におけるヌル終了の使用を制御する 32 ビット整数値。

値 以下の値を指定することができます。

- **SQL_TRUE**: CLI は、ヌル終了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**: CLI は、ヌル終了を出力文字ストリングに使用しません。

この属性の影響を受ける CLI 関数は、文字ストリング・パラメーターのある環境 (およびその環境で割り振られている接続とステートメント) について呼び出されたすべての関数です。

この属性は、この環境に接続ハンドルが割り振られていないときにのみ、設定することができます。

SQL_ATTR_PROCESSCTL

説明 プロセスのすべての環境と接続に影響を与える、プロセス・レベル属性を設定する 32 ビット・マスク。この属性は、環境ハンドルが割り振られる前に設定する必要があります。

SQLSetEnvAttr() の呼び出しでは、 *EnvironmentHandle* 引数を SQL_NULL_HANDLE に設定する必要があります。この設定は、プロセスの所要時間中はずっと有効です。一般に、この属性を使用するのは、大量の CLI 関数呼び出しが行われる、パフォーマンスの影響を受けやすいアプリケーションの場合だけです。以下のビットのいずれかを設定する前に、アプリケーションとアプリケーションが呼び出すその他のライブラリーが、列挙されている制約事項に従っていることを確認してください。

値

下記の値を組み合わせてビット・マスクを形成できます。

- SQL_PROCESSCTL_NOTHREAD - このビットは、アプリケーションが複数のスレッドを使用しないことを示します。あるいは、アプリケーションが複数のスレッドを使用する場合には、アプリケーションによってすべての DB2 呼び出しがシリアライズされます。これを設定すると、CLI は、CLI への呼び出しをシリアライズするためのシステム呼び出しを行わず、DB2 コンテキスト・タイプを SQL_CTX_ORIGINAL に設定します。
- SQL_PROCESSCTL_NOFORK - このビットは、アプリケーションが子プロセスを fork しないことを示します。デフォルトで、CLI はアプリケーションが子プロセスを fork するかどうかを調べません。しかし、CheckForFork CLI/ODBC 構成キーワードが設定されている場合、CLI はキーワードが有効になっているデータベースに接続するすべてのアプリケーションについて、関数呼び出しごとに現行プロセス ID を調べます。CLI がそのアプリケーションの fork されたプロセスを調べないように、この属性を設定できます。

SQL_ATTR_PROCESSCTL 属性は、IBM 定義の拡張機能です。

SQL_ATTR_RESET_CONNECTION

説明 Windows オペレーティング・システムの接続プールに接続が配置されていることを ODBC Driver Manager が ODBC ドライバーに通知するかどうかを指定する 32 ビットの符号なし整数値。SQL_ATTR_ODBC_VERSION 環境属性が SQL_OV_ODBC3_80 に設定されていると、ODBC Driver Manager は、接続プールに接続を配置する前にこの属性を設定するため、ドライバーが他の接続属性をデフォルト値にリセットできます。

値 指定できるのは以下の値のみです。

- SQL_RESET_CONNECTION_YES (デフォルト): ODBC Driver Manager が ODBC ドライバーに接続が接続プールに配置されていることを通知します。

注: SQL_ATTR_RESET_CONNECTION は、ODBC Driver Manager と ODBC ドライバーの間の通信でのみ使用する必要があります。すべての接

続属性がデフォルト値にリセットされるため、この属性をアプリケーションから設定しないでください。例えば、SQLSetConnectAttr() 関数を使用して設定した接続属性の値は、CLI のデフォルト値にリセットされ、アプリケーションが予期しない動作をする可能性があります。

SQL_ATTR_SYNC_POINT

この属性は、DB2 データベース・バージョン 8 から使用すべきでない属性となりました。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_TRACE

説明 CLI/ODBC トレース機能をオンにするのに使用される、ヌル終了文字ストリングを指すポインター。

値 ストリングには、CLI キーワード **TRACE** および **TRACEPATHNAME** が含まれていなければなりません。例えば、以下のようにします。

```
"TRACE=1; TRACEPATHNAME=<dir>;"
```

使用上の注意

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_TRACENOHEADER

説明 CLI トレース・ファイルにヘッダー情報が含まれるかどうかを指定する 32 ビット整数値。

値 以下の値を指定することができます。

- **0** - ヘッダー情報が CLI トレース・ファイルに含まれます。
- **1** - ヘッダー情報は CLI トレース・ファイルに含まれません。

SQL_ATTR_TRACENOHEADER 属性は、SQL_NULL_HANDLE または有効な環境ハンドルとともに使用することができます。

SQL_ATTR_USE_2BYTES_OCTET_LENGTH

この属性は、DB2 データベース・バージョン 8 から非推奨の属性となっています。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA

この属性を設定することは、接続属性 SQL_ATTR_DESCRIBE_OUTPUT_LEVEL を 0 に設定することに相当します。SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA は推奨されないため、アプリケーションは現在、接続属性 SQL_ATTR_DESCRIBE_OUTPUT_LEVEL を使用する必要があります。

SQL_ATTR_USER_REGISTRY_NAME

説明 この属性は、サーバー上で ID マッピング・サービスを使用するユーザーを認証する際にのみ使用されます。

値 SQL_ATTR_USER_REGISTRY_NAME 属性は、ID マッピング・レ

ジストリーに名前を付けるユーザー定義ストリングに設定されます。名前のフォーマットは、ID マッピング・サービスに応じて変化します。この属性を指定することにより、提供したユーザー名がこのレジストリーにあることがサーバーに通知されます。

この属性を設定した後、次に通常接続を確立しようとするとき、次にトラステッド接続を確立しようとするとき、または次にトラステッド接続でユーザー ID を切り替えようとするときに、値が使用されます。

使用上の注意

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

SQL_CONNECTTYPE
この *Attribute* は、SQL_ATTR_CONNECTTYPE に置き換わります。

SQL_MAXCONN
この *Attribute* は、SQL_ATTR_MAXCONN に置き換わります。

SQL_SYNC_POINT
この *Attribute* は、SQL_ATTR_SYNC_POINT に置き換わります。

Informix データベース・サーバーにアクセスする場合、この属性はサポートされません。

接続属性 (CLI) リスト

次の表は、いつそれぞれの CLI 接続属性を設定できるかを示しています。「ステートメントを割り当てた後」列内に「可」があれば、ステートメントの割り振りの前でも後でも接続属性を設定できることを意味します。

表 167. 接続属性をいつ設定するか

属性	接続前	接続後	ステートメントを割り当てた後
SQL_ATTR_ACCESS_MODE	可	可	可 [a]
SQL_ATTR_ALLOW_INTERLEAVED_GETDATA	可	可	可
SQL_ATTR_ANSI_APP	可	不可	不可
SQL_ATTR_APP_USES_LOB_LOCATOR	可	可	可[c]
SQL_ATTR_APPEND_FOR_FETCH_ONLY	可	可	不可
SQL_ATTR_ASYNC_ENABLE	可	可	可 [a]
SQL_ATTR_AUTO_IPD (読み取り専用)	不可	不可	不可
SQL_ATTR_AUTOCOMMIT	可	可	可 [b]
SQL_ATTR_CLIENT_CODEPAGE	可	不可	不可
SQL_ATTR_COLUMNWISE_MRI	可	可	可 [a]
SQL_ATTR_COMMITONEOF	可	可	不可
SQL_ATTR_CONCURRENT_ACCESS_RESOLUTION	可	可	可 [a]
SQL_ATTR_CONN_CONTEXT	可	不可	不可
SQL_ATTR_CONNECT_NODE	可	不可	不可
SQL_ATTR_CONNECTION_DEAD (読み取り専用)	不可	不可	不可
SQL_ATTR_CONNECTTYPE	可	不可	不可
SQL_ATTR_CURRENT_CATALOG (読み取り専用)	不可	不可	不可
SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION	可	可	可
SQL_ATTR_CURRENT_PACKAGE_PATH	可	可	可

表 167. 接続属性をいつ設定するか (続き)

属性	接続前	接続後	ステートメントを割り当てた後
SQL_ATTR_CURRENT_PACKAGE_SET	可	可 [a]	不可 [*]
SQL_ATTR_CURRENT_SCHEMA	可	可	可
SQL_ATTR_DB2_APPLICATION_HANDLE (読み取り専用)	不可	不可	不可
SQL_ATTR_DB2_APPLICATION_ID (読み取り専用)	不可	不可	不可
SQL_ATTR_DB2_SQLERRP (読み取り専用)	不可	不可	不可
SQL_ATTR_DB2EXPLAIN	不可	可	可
SQL_ATTR_DECFLOAT_ROUNDING_MODE	可	可	可
SQL_ATTR_DESCRIBE_CALL	可	可	可 [a]
SQL_ATTR_DESCRIBE_OUTPUT_LEVEL	可	可	不可
SQL_ATTR_ENLIST_IN_DTC	不可	可	可
SQL_ATTR_EXTENDED_INDICATORS	不可	可	可
SQL_ATTR_FET_BUF_SIZE	可	不可	不可
SQL_ATTR_FREE_LOCATORS_ON_FETCH	可	可	可
SQL_ATTR_FORCE_ROLLBACK	可	可	可
SQL_ATTR_GET_LATEST_MEMBER	不可	可	可
SQL_ATTR_INFO_ACCTSTR	不可	可	可
SQL_ATTR_INFO_APPLNAME	不可	可	可
SQL_ATTR_INFO_PROGRAMID	不可	可	可 [a]
SQL_ATTR_INFO_PROGRAMNAME	可	不可	不可
SQL_ATTR_INFO_USERID	不可	可	可
SQL_ATTR_INFO_WRKSTNNAME	不可	可	可
SQL_ATTR_KEEP_DYNAMIC	不可	可	可
SQL_ATTR_LOB_CACHE_SIZE	可	可	可[c]
SQL_ATTR_LOGIN_TIMEOUT	可	不可	不可
SQL_ATTR_LONGDATA_COMPAT	可	可	可
SQL_ATTR_MAX_LOB_BLOCK_SIZE	可	可	可[c]
SQL_ATTR_MAPCHAR	可	可	可
SQL_ATTR_NETWORK_STATISTICS	可	可	可
SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE	不可	可	不可
SQL_ATTR_OVERRIDE_CODEPAGE	不可	可	不可
SQL_ATTR_PARC_BATCH	可	可	可[*]
SQL_ATTR_PING_DB (読み取り専用)	不可	不可	不可
SQL_ATTR_PING_NTIMES	可	可	可
SQL_ATTR_PING_REQUEST_PACKET_SIZE	可	可	可
SQL_ATTR_QUIET_MODE	可	可	可
SQL_ATTR_RECEIVE_TIMEOUT	可	可	可
SQL_ATTR_REOPT	不可	可	可[c]
SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB	可	可	可
SQL_ATTR_REPORT_SEAMLESSFAILOVER_WARNING	可	可	可[*]
SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN	可	可	可
SQL_ATTR_RETRYONERROR	可	可	可
SQL_ATTR_SERVER_MSGTXT_MASK	可	可	可
SQL_ATTR_SERVER_MSGTXT_SP	可	可	可
SQL_ATTR_SESSION_TIME_ZONE	可	不可	不可
SQL_ATTR_SQLCODEMAP	可	不可	不可
SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB	可	可	可
SQL_ATTR_STMT_CONCENTRATOR	可	可	可
SQL_ATTR_STREAM_GETDATA	可	可	可[c]
SQL_ATTR_TRUSTED_CONTEXT_PASSWORD	不可	可	可
SQL_ATTR_TRUSTED_CONTEXT_USERID	不可	可	可
SQL_ATTR_TXN_ISOLATION	不可	可 [b]	可 [a]
SQL_ATTR_USE_TRUSTED_CONTEXT	可	不可	不可

表 167. 接続属性をいつ設定するか (続き)

属性	接続前	接続後	ステートメントを割り当てた後
SQL_ATTR_USER_REGISTRY_NAME	可	不可	不可
SQL_ATTR_WCHARTYPE	可	可 [b]	可 [b]
SQL_ATTR_XML_DECLARATION	可	可	可 [a]

[a] 結果的に割り当てられたステートメントにのみ影響を与えます。

[b] 属性を設定できるのは、その接続上にオープン・トランザクションがない場合だけです。

[c] 属性を設定できるのは、その接続上にオープン・カーソルがない場合だけです。属性はすべてのステートメントに影響を与えます。

[*] ステートメントが割り振られた後でこの属性を設定してもエラーにはなりませんが、どのパッケージがどのステートメントで使用されるかの判断があいまいで、予期しない動作になる可能性があります。この属性を、ステートメントの割り振り後に設定することは、お勧めできません。

属性　ValuePtr の内容

SQL_ATTR_ACCESS_MODE

以下のいずれかである 32 ビット整数値。

- SQL_MODE_READ_ONLY: アプリケーションは、この時点からデータに関する更新を行わないことを示します。したがって、非コミット読み取り (SQL_TXN_READ_UNCOMMITTED) といった、制限の少ない分離レベルおよびロックをトランザクションで使えるようになります。 CLI は、データベースに対する要求が*読み取り専用* であることを確認しません。更新要求を出すと、 CLI は SQL_MODE_READ_ONLY 設定値の結果として選択されたトランザクション分離レベルを使用して、その要求を処理します。
- **SQL_MODE_READ_WRITE (デフォルト)**: アプリケーションは、この時点からデータに関する更新を行うことを示します。 CLI は、この接続に関するデフォルト・トランザクション分離レベルを使用する状態に戻ります。

この接続に未解決のトランザクションがあってはなりません。

SQL_ATTR_ALLOW_INTERLEAVED_GETDATA

アプリケーションが、Dynamic Data Format をサポートするデータ・サーバーを照会するときに、以前にアクセスした LOB 列に対して `SQLGetData()` を呼び出し、 `SQLGetData()` への以前の呼び出し時のデータ・オフセット位置を維持できるかどうかを指定します。この属性の値は次のうちの 1 つとなります。

- SQL_ALLOW_INTERLEAVED_GETDATA_OFF - デフォルト設定は、アプリケーションが以前にアクセスした LOB 列に対して `SQLGetData()` を呼び出すことを許可しません。
- SQL_ALLOW_INTERLEAVED_GETDATA_ON - このキーワードは、Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするデータベース・サーバーへの接続にのみ影響を与えます。このオプションを指定すると、アプリケーションは以前にアクセスした

LOB 列に対して SQLGetData() を呼び出し、以前の読み取り時にアプリケーションが読み取りを停止したところから LOB データの読み取りを開始することができます。

364 ページの『AllowInterleavedGetData CLI/ODBC 構成キーワード』 の設定は、この動作を接続レベルで指定する別の方法です。

SQL_ATTR_ANSI_APP

アプリケーションを ANSI または Unicode アプリケーションとして識別する 32 ビットの符号なし整数。この属性の値は次のうちの 1 つとなります。

- **SQL_AA_TRUE (デフォルト)** : アプリケーションは ANSI アプリケーションです。すべての文字データは、ANSI バージョンの CLI/ODBC 関数を使用して、ネイティブ・アプリケーション (クライアント) のコード・ページでアプリケーションとやり取りされます。
- SQL_AA_FALSE: アプリケーションは Unicode アプリケーションです。Unicode (W) バージョンの CLI/ODBC 関数が呼び出されると、すべての文字データは Unicode でアプリケーションとやり取りされます。

SQL_ATTR_APP_USES_LOB_LOCATOR

アプリケーションが LOB ロケーターを使用するかどうかを示す 32 ビット符号なし整数。この属性の値は次のうちの 1 つとなります。

- **1 (デフォルト)**: アプリケーションが LOB ロケーターを使用するかどうかを示します。
- 0: LOB ロケーターを使用しないアプリケーションで Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするサーバー上のデータを照会している場合は、0 を指定することによって、LOB ロケーターが使用されないことを示し、LOB データの戻りが最適化されるようにしてください。

ストアード・プロシージャーの結果セットの場合、このキーワードは無視されます。

キーワードが 0 に設定されている場合、アプリケーションが SQLBindCol() を使用して LOB ロケーターを結果セットにバインドすると、SQLFetch() 関数から無効な変換エラーが戻されます。

366 ページの『AppUsesLOBLocator CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_APPEND_FOR_FETCH_ONLY

デフォルトで、CLI は、DB2 for z/OS または IBM DB2 for IBM i (DB2 for i) データベースに接続する際に、読み取り SELECT ステートメントに、"FOR FETCH ONLY" 節を追加します。

アプリケーションでこの属性を使用することにより、CLI が "FOR FETCH ONLY" 節をどんな場合に追加するかを接続レベルで制御することができます。例えば、アプリケーションはさまざまなバインド BLOCKING オプション (例えば、BLOCKING UNAMBIG) を使用して CLI パッケージをバインドし、特定の行での位置を維持するために、ブロッキングを抑制します。

デフォルトの CLI の動作を変更するには、キーワードを以下のように設定します。

- 0: CLI は、接続先のサーバー・タイプには関係なく、読み取り SELECT ステートメントに FOR FETCH ONLY 節を追加しません。
- 1: CLI は、接続先のサーバー・タイプには関係なく、読み取り SELECT ステートメントに FOR FETCH ONLY 節を追加します。

この属性は、接続が割り振られた後か、接続が確立された直後に設定しなければなりません。また、アプリケーションの実行期間中に一度設定する必要があります。接続が確立された後かこの属性が設定された後に、アプリケーションは SQLGetConnectAttr() で属性を照会できます。

367 ページの『AppendForFetchOnly CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_ASYNC_ENABLE

指定した接続上のステートメントで呼び出される関数が非同期で実行されるかどうかを指定する 32 ビット整数値。

- **SQL_ASYNC_ENABLE_OFF (デフォルト) = Off**
- SQL_ASYNC_ENABLE_ON = On

SQL_ASYNC_ENABLE_ON を設定すると、この接続で割り振られたすべてのステートメント・ハンドルについて非同期実行が可能になります。接続時にアクティブ・ステートメントがあるときに非同期実行をオンにすると、エラーが戻されます。

この属性では、*InfoType* SQL_ASYNC_MODE とともに呼び出された SQLGetInfo() が SQL_AM_CONNECTION と SQL_AM_STATEMENT のどちらを戻すのかを設定できます。

関数が非同期に呼び出されると、元の関数が SQL_STILL_EXECUTING 以外のコードを戻すまでは、元の関数、SQLAllocHandle()、SQLCancel()、SQLGetDiagField()、または SQLGetDiagRec() だけが、*StatementHandle* に関連したステートメントまたは接続で呼び出せます。 *StatementHandle* または *StatementHandle* に関連した接続で他の関数を呼び出すと、SQL_ERROR が SQLSTATE HY010 (関数シーケンス・エラー) を伴って戻されます。

以下の関数は、非同期に実行できます。SQLBulkOperations()、SQLColAttribute()、SQLColumnPrivileges()、SQLColumns()、SQLDescribeCol()、SQLDescribeParam()、SQLExecDirect()、SQLExecute()、SQLExtendedFetch()、SQLExtendedPrepare()、SQLFetch()、SQLFetchScroll()、SQLForeignKeys()、SQLGetData()、SQLGetLength()、SQLGetPosition()、SQLMoreResults()、SQLNumResultCols()、SQLParamData()、SQLPrepare()、SQLPrimaryKeys()、SQLProcedureColumns()、SQLProcedures()、SQLRowCount()、SQLSetPos()、SQLSpecialColumns()、SQLStatistics()、SQLTablePrivileges()、SQLTables()。

注: Unicode 等価関数も非同期で呼び出すことができます。

SQL_ATTR_AUTO_IPD

SQLPrepare() 呼び出しの後で IPD の自動移植をサポートするかどうかを指定する 32 ビットの符号なしの読み取り専用整数値。

- SQL_TRUE = SQLPrepare() 呼び出し後の IPD の自動移植がサーバーによりサポートされています。

- SQL_FALSE = SQLPrepare() 呼び出し後の IPD の自動移植はサーバーによりサポートされていません。準備状態のステートメントをサポートしないサーバーでは、 IPD を自動的に移植することはできません。

SQL_ATTR_AUTO_IPD 接続属性に SQL_TRUE が戻される場合は、接続属性 SQL_ATTR_ENABLE_AUTO_IPD を設定して、 IPD 自動移植のオン/オフを切り換えることができます。 SQL_ATTR_AUTO_IPD が SQL_FALSE の場合、 SQL_ATTR_ENABLE_AUTO_IPD を SQL_TRUE に設定することはできません。

SQL_ATTR_ENABLE_AUTO_IPD のデフォルト値は、SQL_ATTR_AUTO_IPD の値と等しくなります。

この接続属性は、SQLGetConnectAttr() によって戻せますが、SQLSetConnectAttr() で設定することはできません。

SQL_ATTR_AUTOCOMMIT

自動コミット・モードまたは手動コミット・モードのどちらを使用するかを指定する 32 ビットの符号なしの整数値。

- SQL_AUTOCOMMIT_OFF: アプリケーションは、 SQLEndTran() 呼び出しでトランザクションを手動で明示的にコミットまたはロールバックします。
- **SQL_AUTOCOMMIT_ON (デフォルト)**: デフォルトでは、CLI は自動コミット・モードで動作します。各ステートメントは、暗黙コミットされます。照会ではないそれぞれのステートメントは、実行されるとすぐコミットされるか、または障害発生後にロールバックされます。各照会は、関連付けられているカーソルがクローズするとすぐにコミットされます。

  注: これが調整された分散作業単位の接続である場合、デフォルト値は **SQL_AUTOCOMMIT_OFF** になります。

多くの DB2 環境では、 SQL ステートメントの実行およびコミットは別個にデータベース・サーバーへ流される場合があるので、自動コミットは費用がかかることがあります。アプリケーション開発者が自動コミット・モードを選択するときに、このことを考慮に入れることをお勧めします。

注: 手動のコミット・モードから自動コミット・モードへ変更すると、接続上のオープン・トランザクションをコミットします。

SQL_ATTR_CLIENT_CODEPAGE

ユーザーが CLI アプリケーションから接続レベル・コード・ページを指定できるようにする接続レベル属性値。この属性を指定すると、環境レベルのデフォルト・コード・ページ設定値がオーバーライドされます。

例 1: このデータベース接続が使用するコード・ページを設定する

```
Unicode = 1208;
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_CLIENT_CODEPAGE,
                          (SQLPOINTER)&unicode,  SQL_NTS);
char *connStr = "DSN=EBCDICDB;";
cliRC = SQLDriverConnect (hdbc, (SQLHWND)NULL, connStr, SQL_NTS, NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
```

例 2: SQL_ATTR_CLIENT_CODEPAGE の現行値を取得する

```
cliRC = SQLConnect(hdbc,
                           (SQLCHAR *)"SAMPLE",
                            SQL_NTS,
```

```
                                (SQLCHAR *)"USER1",
                                SQL_NTS,
                                (SQLCHAR *)"PASSWD1",
                                SQL_NTS);

cliRC = SQLGetConnectAttr(hdbc,
                          SQL_ATTR_CLIENT_CODEPAGE,
                          &codePage, 0, NULL);
```

SQL_ATTR_CLIENT_LOB_BUFFERING

バインドされていない LOB 列について、 LOB ロケーターまたはその基になる LOB データが結果セットに入れて戻されるかどうかを指定します。デフォルトでは、ロケーターが戻されます。バインドされていない LOB をアプリケーションがフェッチしてから、その基になる LOB データを取り出すことが通常必要になるのであれば、初めから LOB データを取り出すようにすることによって、アプリケーションのパフォーマンスが改善されることがあります。そのようにするなら、同期待機数やネットワーク・フローが少なくなります。この属性に指定可能な値は、以下のとおりです。

- SQL_CLIENTLOB_USE_LOCATORS (デフォルト) - LOB ロケーターが戻されます。
- SQL_CLIENTLOB_BUFFER_UNBOUND_LOBS - 実際の LOB データが戻されます。

SQL_ATTR_CLIENT_TIME_ZONE

タイム・ゾーン情報が含まれる、±hh:mm の形式でのヌル終了文字ストリング。この属性を指定すると、クライアント・ホストのデフォルトのオペレーティング・システム・タイム・ゾーン値がオーバーライドされます。

SQL_ATTR_COLUMNWISE_MRI

DB2 for z/OS サーバーに接続されている CLI アプリケーションが配列入力チェーニングを INSERT 操作の列方向配列入力に変換できるようにする 32 ビットの符号なし整数。この属性は、バージョン 9.7 フィックスパック 5 以降で使用可能です。指定可能な値は次のとおりです。

- **SQL_COLUMNWISE_MRI_OFF (デフォルト)**:CLI は、チェーニング・データを列方向配列入力に変換しません。
- SQL_COLUMNWISE_MRI_ON: CLI は、配列入力チェーニングを列方向配列入力に変換します。DB2 for z/OS の複数行挿入 (MRI) フィーチャーは、データが列方向配列の形式であることを想定しています。アプリケーションが配列入力チェーニングを使用している場合、この変換によって、SQLExecute () を呼び出すたびにデータが圧縮された配列の形式で送信されるため、アプリケーションのパフォーマンスを最適化するのに役立ちます。配列入力チェーニングについて詳しくは、SQL_ATTR_CHAINING_BEGIN を参照してください。

DB2 for z/OS サーバー以外では、CLI は自動的にチェーニング・データを行方向配列入力に変換するため、この属性を設定しても効果はありません。

次の場合には、変換は実行されません。

- パラメーターを SQL_CLOB、SQL_BLOB、SQL_LONGVARBINARY、SQL_LONGVARGRAPHIC、SQL_DBCLOB、SQL_XML などの LOB データ・タイプにバインドする場合。

- SQLPutData() 関数および SQLParamData() 関数を呼び出して、値を SQL_DATA_AT_EXEC に設定してデータを INSERT 操作に渡すことによって、入力される実行時データ・パラメーターをバインドする場合。
- 内部バッファー内ですべてのアプリケーション・データを格納するためのスペースを利用できない場合。

SQL_ATTR_COMMITONEOF

結果セット全体を読み取り、EOF を受け取った直後に、暗黙的な COMMIT を実行するかどうかを指定する 32 ビットの整数値。この接続属性は、バージョン 9.7 フィックスパック 5 以降で使用可能です。この属性の指定可能な値は次のとおりです。

- **SQL_ COMMITONEOF_OFF (デフォルト)**: 結果セット全体を読み取った後で、COMMIT は暗黙的に実行されません。カーソルを閉じてリソースを解放するには、明示的に SQLFreeStmt() 関数を呼び出す必要があります。
- SQL_ COMMITONEOF_ON: 結果セット全体を読み取った後で、暗黙的な COMMIT が実行されます。

注: この属性を使用しても、`SQLFreeStmt()` 関数への必要な呼び出しを置き換えることはありません。

SQL_ATTR_CONCURRENT_ACCESS_RESOLUTION

ステートメント・レベルで使用する並行アクセス解決方法を指定する 32 ビット整数値。 この設定は、カーソル固定 (CS) スキャンに対して指定されているデフォルト動作をオーバーライドします。

- 0 = 設定なし。クライアントは準備オプションを提供しません。
- 1 = Currently Committed セマンティクスを使用します。CLI は、準備が行われる度に「Currently Committed」をコマンド・フローの一部として送信します。つまり、データベース・マネージャーは、データが更新または削除の処理中であっても、該当するスキャンについてデータの現在コミット済みバージョンを使用することができます。挿入処理中の行はスキップすることができます。この設定は、実際の分離レベルがカーソル固定または読み取り固定のときに適用され (読み取り固定の場合はコミットされていない挿入のみをスキップ)、それ以外のときには無視されます。該当するスキャンには、読み取り専用スキャン (読み取り専用ステートメントの一部であることも、読み取り専用でないステートメントの一部であることもある) が含まれます。レジストリー変数 **DB2_EVALUNCOMMITTED**、**DB2_SKIPDELETED**、および **DB2_SKIPINSERTED** の設定は、currently committed を使用するスキャンには適用されません。ただし、これらのレジストリー変数の設定は、currently committed を使用しないスキャンには引き続き適用されます。
- 2 = 結果を待機します。CLI は、準備が行われる度に「結果の待機」をコマンド・フローの一部として送信します。つまり、カーソル固定以上のスキャンは、更新または削除の処理中のデータを検出した場合にコミットまたはロールバックを待機します。挿入処理中の行はスキップされません。レジストリー変数 DB2_EVALUNCOMMITTED、DB2_SKIPDELETED、および DB2_SKIPINSERTED の設定は適用されなくなりました。

- 3 = ロックされたデータをスキップします。CLI は、準備が行われる度に「ロックされたデータをスキップ」をコマンド・フローの一部として送信します。つまり、Currently Committed セマンティクスが使用され、挿入処理中の行はスキップされます。このオプションは、DB2 Database for Linux, UNIX, and Windows ではサポートされません。指定されていても、この設定は無視されます。

DB2 Database for Linux, UNIX, and Windows の場合、この属性を使用して、**cur_commit** 構成パラメーターによって定義された Currently Committed のデフォルトの動作をオーバーライドします。 DB2 for z/OS の場合、この属性を使用して、Currently Committed の動作を有効にします。この動作を指定することに相当する、DB2 for z/OS で使用可能なデータベース構成パラメーターはありません。

381 ページの『ConcurrentAccessResolution CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_CONN_CONTEXT

接続でどのコンテキストを使用するかを示します。 SQLPOINTER は次のいずれかになります。

- コンテキストを設定する、有効なコンテキスト (`sqleBeginCtx()` DB2 API によって割り当てられているもの)。
- コンテキストをリセットする NULL ポインター。

この属性を使用できるのは、アプリケーションが DB2 コンテキスト API を使ってマルチスレッド・アプリケーションを管理している場合だけです。デフォルト設定では、CLI は接続ハンドルごとに 1 つのコンテキストを割り当て、実行スレッドが確実に正しいコンテキストにアタッチされるようにすることにより、コンテキストを管理しています。

コンテキストの詳細については、`sqleBeginCtx()` API を参照してください。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CONNECT_NODE

接続先の DB2 Enterprise Server Edition データベース・パーティション・サーバーのターゲット論理パーティションを指定する 32 ビット整数。この属性に指定可能な値は、以下のとおりです。

- 0 から 999 までの整数
- SQL_CONN_CATALOG_NODE

この変数を設定しない場合のデフォルトのターゲット論理ノードは、マシン上でポート 0 と定義された論理ノードになります。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

また、対応するキーワード、382 ページの『ConnectNode CLI/ODBC 構成キーワード』があります。

SQL_ATTR_CONNECTION_DEAD

接続が依然としてアクティブであるかどうかを示す、読み取り専用 32 ビット整数値。 CLI は、次のいずれかの値を戻します。

- SQL_CD_FALSE - 接続はアクティブのままです。
- SQL_CD_TRUE - エラーが起きたので、サーバーへの接続が終了されました。この場合でもアプリケーションは切断を実行して、すべての CLI リソースの終結処理を行う必要があります。

この属性が主に使用されるのは、接続をプールする前の Microsoft ODBC Driver Manager 3.5x においてです。

SQL_ATTR_CONNECTION_TIMEOUT

この接続属性は ODBC で定義されていますが、CLI ではサポートされません。この属性を設定または取得しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_CONNECTTYPE

このアプリケーションを整合分散環境で実行するか、それとも非整合分散環境で実行するかを指定する 32 ビット整数値。可能な値は次のとおりです。

- **SQL_CONCURRENT_TRANS (デフォルト)**: アプリケーションを使用して、1 つ以上のデータベースへの並行複数接続を行うことができます。各接続には、それぞれのコミット範囲があります。トランザクションの調整の実施は試みられません。あるアプリケーションが SQLEndTran() 上の環境ハンドルを使用してコミットを発行したが、すべての接続コミットが成功したわけではない場合、そのアプリケーションはリカバリーを行う必要があります。
- SQL_COORDINATED_TRANS: アプリケーションはコミットを行い、複数のデータベース接続で調整をロールバックします。このオプション設定は、組み込み SQL のタイプ 2 CONNECT の指定に対応しています。SQL_CONCURRENT_TRANS 設定とは対照的に、アプリケーションは 1 つのデータベースにつき 1 つのオープン接続のみを許可されます。

  注: この接続タイプでは、 SQL_ATTR_AUTOCOMMIT 接続オプションのデフォルト値である SQL_AUTOCOMMIT_OFF の設定になります。

この属性をデフォルトから変更する場合、接続を環境ハンドルに対して確立する前にこれを設定する必要があります。

環境ハンドルが割り振られたら、必要に応じて、アプリケーションはできる限り即時に SQLSetEnvAttr() への呼び出しを行って、この属性を環境属性として設定することが推奨されています。しかし、ODBC アプリケーションは SQLSetEnvAttr() にアクセスできないので、個々の接続ハンドルが割り振られてから接続が確立されるまでの間に、SQLSetConnectAttr() を使用してこの属性を設定しなければなりません。

環境ハンドル上のすべての接続の SQL_ATTR_CONNECTTYPE 設定は、同じでなければなりません。環境では、並行接続と整合接続を混合して使うことはできません。最初の接続のタイプが決まると、それ以降のすべての接続のタイプはそれに従います。 SQLSetEnvAttr() は、接続アクティブに接続タイプを変更しようとすると、エラーが返されます。

384 ページの『ConnectType CLI/ODBC 構成キーワード』を使用して、このデフォルト接続タイプを設定することもできます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_CURRENT_CATALOG
データ・ソースで使用するカタログの名前が入れられるヌル終了文字ストリング。多くの場合、カタログ名はデータベース名と同じです。

この接続属性は、SQLGetConnectAttr() によって戻せますが、SQLSetConnectAttr() で設定することはできません。この属性を設定しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION
CURRENT IMPLICIT XMLPARSE OPTION 特殊レジスターを設定するために使用されるストリング定数である、ヌル終了文字ストリング。この属性を設定すると、 SET CURRENT IMPLICIT XMLPARSE OPTION SQL ステートメントが発行されます。この属性が接続の確立前に設定された場合、接続時に SET CURRENT IMPLICIT XMLPARSE OPTION SQL ステートメントが発行されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CURRENT_PACKAGE_PATH
複数のパッケージが構成されている場合にパッケージを解決するために、DB2 データベース・サーバーが使用するパッケージ修飾子のヌル終了文字ストリング。この属性を設定すると、データベース・サーバーに接続するたびに "SET CURRENT PACKAGE PATH = *schema1, schema2, ...*" ステートメントが発行されます。

この属性は、 CLI アプリケーションではなく ODBC 静的処理アプリケーションで使用することに最も適しています。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_CURRENT_PACKAGE_SET

後続の SQL ステートメント用のパッケージの選択に使用されるスキーマ名 (コレクション ID) を示すヌル終了文字ストリング。この属性を設定すると、 SET CURRENT PACKAGESET SQL ステートメントが発行されます。この属性が接続以前に設定された場合、接続時に SET CURRENT PACKAGESET SQL ステートメントが発行されます。

CLI/ODBC アプリケーションは動的 SQL ステートメントを発行します。この接続属性を使用すると、これらのステートメントの実行に使用される特権をコントロールできます。

- CLI/ODBC アプリケーションから SQL ステートメントを実行するときに使用するスキーマを選択します。
- スキーマ内のオブジェクトに必要な特権があることを確認してから、それに従って再バインドします。これは特に、COLLECTION <collid> オプシ

ョンを使用して、 CLI パッケージ (sqllib/bnd/db2cli.lst) をバインドすることを意味します。詳細については、BIND コマンドを参照してください。

- CURRENTPACKAGESET オプションをこのスキーマに設定します。

これで CLI/ODBC アプリケーションからの SQL ステートメントが、指定したスキーマの下で実行され、そこに定義されている特権を使用します。

387 ページの『CurrentPackageSet CLI/ODBC 構成キーワード』 の設定は、スキーマ名を指定する別の方法です。

次のパッケージ・セット名は予約済みです。
NULLID、NULLIDR1、NULLIDRA。

SQL_ATTR_REOPT と SQL_ATTR_CURRENT_PACKAGE_SET を同時に指定することはできません。したがって、一方を設定すると、他方は許可されません。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CURRENT_SCHEMA
szSchemaName ポインターを NULL に設定する場合に、 SQLColumns() 呼び出しで CLI が使用するスキーマの名前の入ったヌル終了文字ストリング。

このオプションをリセットするには、 *ValuePtr* 引数で長さゼロのストリングまたは NULL ポインターを使ってこのオプションを指定します。

このオプションは、アプリケーション開発者が、 SQLColumns() への一般呼び出しを次のようにコーディングしてある場合に便利です。つまり、スキーマ名で結果セットを制限せず、結果セットをコード内で孤立させて制約をかける必要がある場合です。

このオプションはいつでも設定することができますが、 *szSchemaName* ポインターが NULL である次の SQLColumns() 呼び出しで有効になります。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_DB2_APPLICATION_HANDLE
接続のアプリケーション・ハンドルを戻すユーザー定義文字ストリング。アプリケーション・ハンドル全体を含めるだけの大きさがストリングにない場合は、切り捨てられます。

この接続属性は、SQLGetConnectAttr() によって戻せますが、SQLSetConnectAttr() で設定することはできません。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_DB2_APPLICATION_ID
接続のアプリケーション ID を戻すユーザー定義文字ストリング。アプリケーション ID 全体を含めるだけの大きさがストリングにない場合は、切り捨てられます。

この接続属性は、SQLGetConnectAttr() によって戻せますが、SQLSetConnectAttr() で設定することはできません。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_DB2_SQLERRP

sqlca の *sqlerrp* フィールドを収めたヌル終了ストリングを指す sqlpointer。

製品を示す 3 文字の ID の後に、製品のバージョン、リリース、および修正レベルを示す 5 桁の英数字が続きます。 A-Z の文字はレベル 9 より上の修正レベルを示しています。A は修正レベル 10 を、B は修正レベル 11 を示す、という具合です。 例えば、SQL0907C とは、DB2 バージョン 9 リリース 7 修正レベル 12 を示しています。

SQLCODE がエラー条件を示している場合は、このフィールドはエラーを戻したモジュールを識別します。

接続が正しく完了したときにも、このフィールドが使用されます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_DB2ESTIMATE

この属性は、DB2 UDB バージョン 8 から使用すべきでない属性となりました。

SQL_ATTR_DB2EXPLAIN

サーバーで Explain スナップショットまたは Explain モード情報 (あるいは両方) を生成するかどうかを指定する 32 ビット整数。指定できる値は以下のとおりです。

- SQL_DB2EXPLAIN_OFF: Explain スナップショット機能も Explain 表オプション機能も無効になっています (SET CURRENT EXPLAIN SNAPSHOT=NO と SET CURRENT EXPLAIN MODE=NO がサーバーに送信されます)。
- SQL_DB2EXPLAIN_SNAPSHOT_ON: Explain スナップショット機能は有効、Explain 表オプション機能は無効になっています (SET CURRENT EXPLAIN SNAPSHOT=YES と SET CURRENT EXPLAIN MODE=NO がサーバーに送信されます)。
- SQL_DB2EXPLAIN_MODE_ON: Explain スナップショット機能は有効、Explain 表オプション機能は無効になっています (SET CURRENT EXPLAIN SNAPSHOT=NO と SET CURRENT EXPLAIN MODE=YES がサーバーに送信されます)。
- SQL_DB2EXPLAIN_SNAPSHOT_MODE_ON: Explain スナップショット機能も Explain 表オプション機能も有効になっています (SET CURRENT EXPLAIN SNAPSHOT=YES と SET CURRENT EXPLAIN MODE=YES がサーバーに送信されます)。

Explain 情報を生成する前に、Explain 表を作成する必要があります。

このステートメントはトランザクションに制御されませんし、ROLLBACK の影響も受けません。新規の SQL_ATTR_DB2EXPLAIN 設定は、この接続の次のステートメント準備で有効になります。

現在の許可 ID に、EXPLAIN 表の INSERT 特権が必要です。

391 ページの『DB2Explain CLI/ODBC 構成キーワード』を使用してデフォルト値を設定することもできます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_DECFLOAT_ROUNDING_MODE

10 進数浮動小数点丸めモードは、DECFLOAT の変数または列に入っている値の桁数が DECFLOAT データ・タイプで許可される桁数を超えている場合に、どのタイプの丸めを使用するかを決定します。このようなことは、別のタイプから挿入、更新、選択、変換する際、あるいは数学演算の結果として生じることがあります。

SQL_ATTR_DECFLOAT_ROUNDING_MODE の値は、新しい接続の接続属性によって他のモードが指定されていない場合に、その新しい接続で使用される 10 進数浮動小数点丸めモードを決定します。指定されたどの接続に対しても、CLI および DB2 の両方が、その接続の一部として開始されるすべてのアクションで、同じ 10 進数浮動小数点丸めモードを使用します。

アプリケーションが DB2 Database for Linux, UNIX, and Windows バージョン 9.5 サーバーに接続する場合は、データベース・クライアントの 10 進数浮動小数点丸めモードを、サーバーで設定されているのと同じモードに設定する必要があります。クライアントの 10 進数浮動小数点丸めモードを、データベース・サーバーで設定されている 10 進数浮動小数点丸めモードとは異なる値に設定していると、接続時にデータベース・サーバーから SQL0713N が戻されます。

これらの 10 進数浮動小数点丸めモードに対応する設定値は、次のとおり。

- 0 = Half even (デフォルト)
- 1 = Half up
- 2 = Down
- 3 = Ceiling
- 4 = Floor

各モードは、次のとおりです。

Half even (デフォルト)
このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値に最も近似の数値を使用します。 2 つの数値がほぼ同じ程度の近似値の場合には、偶数であるものを使用します。このモードは、大量のデータに対して、生じる丸め誤差を最も小さくします。

Half up
このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値に最も近似の数値を使用します。 2 つの数値がほぼ同じ程度の近似値の場合には、元の値よりも大きいものを使用します。

Down このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値に最も近似で、その絶対値が元の値の絶対値よりも大きくない数値を使用します。これは、ゼロに向かっての丸め、または負の値の場合の Ceiling の使用、および正の値の場合の Floor の使用とみなすことができます。

Ceiling
このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値より大か等しい数値で最小のものを使用します。

Floor このモードでは、CLI および DB2 は、ターゲット変数に収まり、元の値より小か等しい数値で最大のものを使用します。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_DESCRIBE_CALL
ストアード・プロシージャー引数がいつ記述されるかを示す 32 ビット整数値。デフォルトでは、CALL ステートメントを準備する際に、CLI は入力パラメーター記述情報を要求しません。アプリケーションがパラメーターをステートメントに正常にバインドした場合には、この記述情報は不要になるため、記述情報を要求しないとパフォーマンスが改善されます。オプションの値は、次のとおりです。

- 1 = SQL_DESCRIBE_CALL_BEFORE。
- -1 = SQL_DESCRIBE_CALL_DEFAULT。

397 ページの『DescribeCall CLI/ODBC 構成キーワード』を使用してこの属性を設定することができます。選択可能なオプションの使用法情報および説明については、キーワードを参照してください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_DESCRIBE_OUTPUT_LEVEL
準備要求または記述要求で CLI ドライバーが要求する情報量を制御するヌル終了文字ストリング。デフォルトでは、サーバーが記述要求を受け取ると、結果セット列に 503 ページの表 168 のレベル 2 に含まれる情報を戻します。ただし、アプリケーションがこの情報のすべてを必要としないことも、あるいは追加情報を必要とすることもあります。
SQL_ATTR_DESCRIBE_OUTPUT_LEVEL 属性を、クライアント・アプリケーションの要件に合ったレベルに設定すると、パフォーマンスが改善される可能性があります。なぜなら、クライアントとサーバー間で転送される記述データが、アプリケーションが必要とする最小量に制限されるためです。
SQL_ATTR_DESCRIBE_OUTPUT_LEVEL 設定値が低すぎると、アプリケーションの機能に影響を与えることがあります (アプリケーションの要件によって異なる)。この場合、記述情報を検索する CLI 関数は失敗しないかもしれませんが、戻される情報は不完全である可能性があります。
SQL_ATTR_DESCRIBE_OUTPUT_LEVEL のサポートされる設定値は、次のとおりです。

- 0 - 記述情報は、クライアント・アプリケーションに戻されません。
- 1 - レベル 1 に分類される記述情報 (503 ページの表 168 を参照) が、クライアント・アプリケーションに戻されます。
- 2 - (デフォルト) レベル 2 に分類される記述情報 (503 ページの表 168 を参照) が、クライアント・アプリケーションに戻されます。
- 3 - レベル 3 に分類される記述情報 (503 ページの表 168 を参照) が、クライアント・アプリケーションに戻されます。

以下の表は、サーバーが準備要求または記述要求を受け取ったときに戻す、記述情報を形成するフィールドをリストします。これらのフィールドは各レベルにグループ化され、SQL_ATTR_DESCRIBE_OUTPUT_LEVEL 属性が、CLI ドライバーの要求する記述情報のレベルを制御します。

注:

1. 必ずしもすべての記述情報のレベルが、すべての DB2 サーバーによってサポートされるとは限りません。記述情報のすべてのレベルがサポートされるのは、次の DB2 サーバーにおいてです。DB2 on Linux、UNIX、および Windows バージョン 8 以降、DB2 for z/OS バージョン 8 以降、および DB2 for i バージョン 5 リリース 3 以降。他のすべての DB2 サーバーは、SQL_ATTR_DESCRIBE_OUTPUT_LEVEL に対して 2 または 0 の設定値のみサポートします。
2. デフォルトの動作では、最初にデフォルト・レベル 2 を使用して取り出されなかった記述情報をアプリケーションが要求する場合に、CLI はレベルを 3 にプロモートできます。これにより、サーバーに対する 2 つのネットワークの流れが存在する可能性があります。アプリケーションがこの属性を使用して明示的に記述レベルを設定する場合、プロモーションは行われません。したがって、この属性を使用して記述レベルを 2 に設定する場合、アプリケーションが拡張情報を求めても、CLI はレベル 3 にプロモートしません。

表 168. 記述情報のレベル

レベル 1	レベル 2	レベル 3
SQL_DESC_COUNT SQL_COLUMN_COUNT SQL_DESC_TYPE SQL_DESC_CONCISE_TYPE SQL_COLUMN_LENGTH SQL_DESC_OCTET_LENGTH SQL_DESC_LENGTH SQL_DESC_PRECISION SQL_COLUMN_PRECISION SQL_DESC_SCALE SQL_COLUMN_SCALE SQL_DESC_DISPLAY_SIZE SQL_DESC_NULLABLE SQL_COLUMN_NULLABLE SQL_DESC_UNSIGNED SQL_DESC_SEARCHABLE SQL_DESC_LITERAL_SUFFIX SQL_DESC_LITERAL_PREFIX SQL_DESC_CASE_SENSITIVE SQL_DESC_FIXED_PREC_SCALE	all fields of level 1 and: SQL_DESC_NAME SQL_DESC_LABEL SQL_COLUMN_NAME SQL_DESC_UNNAMED SQL_DESC_TYPE_NAME SQL_DESC_DISTINCT_TYPE SQL_DESC_REFERENCE_TYPE SQL_DESC_STRUCTURED_TYPE SQL_DESC_USER_TYPE SQL_DESC_LOCAL_TYPE_NAME SQL_DESC_USER_DEFINED_ TYPE_CODE	all fields of levels 1 and 2 and: SQL_DESC_BASE_COLUMN_NAME SQL_DESC_UPDATABLE SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_SCHEMA_NAME SQL_DESC_CATALOG_NAME SQL_DESC_TABLE_NAME SQL_DESC_BASE_TABLE_NAME

399 ページの『DescribeOutputLevel CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_ENLIST_IN_DTC

以下のいずれかの値 SQLPOINTER。

- NULL 以外のトランザクション・ポインター: アプリケーションは、接続の状態を非分散トランザクション状態から分散トランザクション状態に変

更するよう、CLI に要求します。接続の参加は、分散トランザクション・コーディネーター (DTC) によって行われます。

- NULL: アプリケーションは、接続の状態を分散トランザクション状態から非分散トランザクション状態に変更するよう、CLI に要求します。

この属性は、Microsoft Transaction Server (MTS) との接続の参加、または参加の取り止めを行うために、MTS の環境でのみ使用されます。

この属性に非 NULL トランザクション・ポインターを指定して使用すると、直前のトランザクションは終了して、新しいトランザクションが開始されたものと想定されます。非 NULL ポインターを指定してこの API を呼び出す前に、アプリケーションは ITransaction メンバー関数 Endtransaction を呼び出す必要があります。そうしないと、直前のトランザクションが打ち切られてしまいます。アプリケーションは、同じトランザクション・ポインターを使用して複数の接続を参加させることができます。

注: この接続属性は、トランザクションごとに MTS が自動的に指定するものであって、ユーザー・アプリケーションがコーディングするものではありません。
CLI/ODBC アプリケーションは、同一のトランザクションに参加する 2 つの異なる接続上で、同一のデータベースに対して複数の SQL ステートメントを並行に実行することはできません。

SQL_ATTR_EXTENDED_INDICATORS

32 ビットの整数で、ユーザーはこの値を使用すると、サポート対象のサーバーから拡張標識フィーチャーを使用できます。拡張標識がサポートされていないデータ・サーバーに対してこの属性をユーザーが設定しようとすると、CLI アプリケーションに該当するエラーが戻ります。この属性で指定できる値は次のとおりです。

- SQL_EXTENDED_INDICATOR_ENABLE: この場合、ユーザーは SQLBindParameter() / SQLExtendedBind() メソッドに SQL_UNASSIGNED と SQL_DEFAULT_PARAM を示す値を指定できます。
- SQL_EXTENDED_INDICATOR_NOT_SET (デフォルト): このフィーチャーはデフォルトで使用不可になっています。
  SQL_ATTR_EXTENDED_INDICATORS を使用してこのフィーチャーを使用可能にする前に SQL_UNASSIGNED と SQL_DEFAULT_PARAM が使用されると、ユーザーは InvalidArgument 値エラー CLI0124E を受け取ります。
- DB2 for Linux, UNIX, and Windows および DB2 10 for z/OS のデータ・サーバーの拡張標識は、DB2 バージョン 9.7 フィックスパック 2 以降でサポートされます。DB2 for IBM i 7.1 のデータ・サーバーの拡張標識は、DB2 バージョン 9.7 フィックスパック 5 以降でサポートされます。

SQL_ATTR_FET_BUF_SIZE

アプリケーションが、デフォルトの照会ブロック・サイズを 64 K から 256 K の範囲の最適値に設定できるようにするための接続レベル属性。この属性は、接続が作成される前に設定してください。また CLI は、db2cli.ini レベルのキーワード FET_BUF_SIZE も提供します。このキーワードは、db2cli.ini ファイルおよび接続ストリングに設定できます。

それに対応する db2dsdriver.cfg キーワードの FetchBufferSize (db2dsdriver.cfg ファイルで設定可能) も使用できます。

SQL_ATTR_FREE_LOCATORS_ON_FETCH
LOB ロケーターが COMMIT の発行時ではなく、SQLFetch() の実行時に解放されるかどうかを指定する Boolean 属性。この属性を 1 (true) に設定すると、アプリケーションが LOB 列を SQLBindCol() (またはそれに相当する記述子 API) にバインドしないで LOB データをフェッチするときに内部的に使用されるロケーターが解放されます。その場合も、アプリケーションに明示的に戻されるロケーターは、アプリケーションによって解放する必要があります。この属性値は、アプリケーションが SQLCODE = -429 (ロケーターがない) を受け取る事態を避けるために使用できます。この属性のデフォルトは 0 (false) です。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_FORCE_ROLLBACK
DB2 for z/OS および DB2 for OS/390 サーバーへの接続に対して、実行時データ・フローの SQLEndTran() 関数への呼び出しを可能にする 32 ビットの符号なし整数値。

実行時データ・フローにおいて、ユーザーのアプリケーションで SQL_ROLLBACK を *CompletionType* と指定したSQLEndTran() 関数を呼び出すには、StreamPutData 構成キーワードを 1 に設定し、SQL_ATTR_FORCE_ROLLBACK 接続属性も設定する必要があります。

DB2 for z/OS および DB2 for OS/390 サーバーではないデータ・サーバーへの接続では、CLI0150E エラー・メッセージが戻されます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_GET_LATEST_MEMBER
これはバージョン 9.7 フィックスパック 3 以降で使用可能な接続レベル属性であり、これを使用すると、特定の接続に使用された (使用されている) 最新のメンバーを CLI アプリケーションで検索できるようになります。

CLI アプリケーションで、 SQLGetConnectAttr() 関数を呼び出すことによって、現在接続されているメンバーまたは前回接続されたメンバーを検索することができます。

データベース接続を確立する前にこの属性の使用を試行すると、CLI0126E エラー・メッセージが戻されます。

SQL_ATTR_INFO_ACCTSTR
DB2 Connect または DB2 Database for Linux, UNIX, and Windows の使用時に、データ・サーバーに送信されるクライアント・アカウンティング・ストリングを識別するのに使用される、 NULL 文字で終了する文字ストリングを指すポインター。

以下の点に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 200 文字までの長さです。

- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_INFO_APPLNAME
DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・アプリケーション名を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

以下の点に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 32 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_INFO_PROGRAMID
アプリケーションと接続を関連付ける最大長 80 バイトのユーザー定義文字ストリング。この属性を設定すると、DB2 UDB for z/OS バージョン 8 以降では、動的 SQL ステートメント・キャッシュに挿入されたステートメントにこの ID が関連付けられます。

この属性は、DB2 UDB for z/OS バージョン 8 以降または IBM Informix Dynamic Servers (IDS) にアクセスする CLI アプリケーションでのみサポートされます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_INFO_PROGRAMNAME
長さが 20 バイト以下のヌル終了ユーザー定義文字ストリング。クライアントで実行されているアプリケーションの名前を指定します。

サーバーとの接続が確立されるより前にこの属性が設定されている場合、ここに指定される値は実際のクライアント・アプリケーション名をオーバーライドし、 appl_name モニター・エレメントに表示される値になります。
DB2 for z/OS サーバーとの接続では、この設定値の最初の 12 文字が、対応する DB2 for z/OS スレッドの CORRELATION IDENTIFIER として使用されます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_INFO_USERID
DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・ユーザー ID を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

以下の点に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 16 文字までの長さです。
- このユーザー ID を認証ユーザー ID と混同しないでください。このユーザー ID は識別のためだけに使用され、許可にはまったく使用されません。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_INFO_WRKSTNNAME

DB2 Connect、または Linux、UNIX、および Windows 用の DB2 データベース製品の使用時に、データ・サーバーに送信されるクライアント・ワークステーション名を識別するのに使用される、ヌル終了文字ストリングを指すポインター。

以下の点に注意してください。

- 値の設定中、サーバーによっては、指定した長さ全体を処理せず、値を切り捨てる場合があります。
- DB2 for z/OS および OS/390 サーバーがサポートするのは、最大 18 文字までの長さです。
- ホスト・システムへの送信時にデータが正確に変換されるようにするには、A から Z まで、 0 から 9 まで、および下線 (_) またはピリオド (.) の文字だけを使用するようにしてください。.

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_KEEP_DYNAMIC

KEEPDYNAMIC オプションが有効かどうかを指定する 32 ビット符号なし整数値。有効な場合、サーバーは、動的に準備されたステートメントを、トランザクション境界を越えて準備済み状態に維持します。

- 0 - KEEPDYNAMIC 機能は利用できません。 CLI パッケージは KEEPDYNAMIC NO オプションを指定してバインドされました。
- 1 - KEEPDYNAMIC 機能を利用できます。 CLI パッケージは KEEPDYNAMIC YES オプションを指定してバインドされました。

この属性が設定されている場合には、
SQL_ATTR_CURRENT_PACKAGE_SET 属性も設定するようお勧めします。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_LOB_CACHE_SIZE

LOB の最大キャッシュ・サイズ (バイト単位) を指定する、32 ビットの符号なし整数。デフォルトでは、LOB はキャッシュに入れられません。

詳しい使用情報については、 411 ページの『LOBCacheSize CLI/ODBC 構成キーワード』を参照してください。

SQL_ATTR_LOGIN_TIMEOUT
試行を終了し通信タイムアウトを生成する前の、サーバーへの接続の確立を試行するときに応答を待機する秒数に相当する 32 ビット整数値。正の整数を、最高 32 767 まで指定してください。デフォルト設定の 0 を指定すると、クライアントは無制限に待機できます。

接続タイムアウト値の設定は、 383 ページの『ConnectTimeout CLI/ODBC 構成キーワード』を使用して行うこともできます。使用法の情報については、キーワードを参照してください。

SQL_ATTR_LONGDATA_COMPAT
既存のアプリケーションでシームレスにラージ・オブジェクト・データ・タイプにアクセスできるようにするため、文字データ・タイプ、2 バイト文字データ・タイプ、およびバイナリー・ラージ・オブジェクト・データ・タイプを、SQL_LONGVARCHAR、SQL_LONGVARGRAPHIC、または SQL_LONGBINARY として報告するかどうかを指定する 32 ビット整数。オプションの値は、次のとおりです。

- **SQL_LD_COMPAT_NO (デフォルト)**: ラージ・オブジェクト・データ・タイプは、IBM 定義のタイプ (SQL_BLOB、SQL_CLOB、SQL_DBCLOB) で報告されます。
- SQL_LD_COMPAT_YES: IBM ラージ・オブジェクト・データ・タイプ (SQL_BLOB、SQL_CLOB、および SQL_DBCLOB) は、SQL_LONGVARBINARY、SQL_LONGVARCHAR、および SQL_LONGVARGRAPHIC にマップされます。 `SQLGetTypeInfo()` は、SQL_LONGVARBINARY SQL_LONGVARCHAR、および SQL_LONGVARGRAPHIC にそれぞれ 1 つの項目を戻します。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_MAPCHAR
SQL_CHAR、SQL_VARCHAR、SQL_LONGVARCHAR に関連したデフォルトの SQL タイプを指定するために使用する 32 ビット整数値。オプションの値は、次のとおりです。

- **SQL_MAPCHAR_DEFAULT (デフォルト)**: デフォルトの SQL タイプ表記を戻します。
- SQL_MAPCHAR_WCHAR: SQL_WCHAR として SQL_CHAR、SQL_WVARCHAR として SQL_VARCHAR、および SQL_WLONGVARCHAR として SQL_LONGVARCHAR を戻します。

この属性の設定値は、次の CLI 関数にのみ影響します。

- SQLColumns()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()

- SQLProcedureColumns()

SQL_CHAR、SQL_VARCHAR、SQL_LONGVARCHAR に関連したデフォルトの SQL タイプの設定は、 414 ページの『MapCharToWChar CLI/ODBC 構成キーワード』を使用して行うこともできます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_MAXCONN
この属性は、DB2 UDB バージョン 8 から使用すべきでない属性となりました。

SQL_ATTR_MAX_LOB_BLOCK_SIZE
LOB または XML データ・ブロックの最大サイズを示す 32 ビット符号なし整数。正の整数を、最高 2 147 483 647 まで指定してください。デフォルト設定の 0 は、LOB または XML データ・ブロックのデータ・ブロック・サイズに制限がないことを示します。

データ検索時にサーバーは、最大ブロック・サイズに到達している場合でも、現在行に関するすべての情報をクライアントへの応答に含めます。

MaxLOBBlockSize と db2set レジストリー変数 DB2_MAX_LOB_BLOCK_SIZE の両方が指定されている場合、MaxLOBBlockSize の値が使用されます。

422 ページの『MaxLOBBlockSize CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_METADATA_ID
この接続属性は ODBC で定義されていますが、CLI ではサポートされません。この属性を設定または取得しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_NETWORK_STATISTICS
バージョン 9.7 フィックスパック 3 以降、この 32 ビット整数の接続属性は、CLI が接続のネットワーク統計を収集するかどうかを制御します。アプリケーションは、SQLGetDiagField() 関数を呼び出し、*DiagIdentifier* 引数に対して SQL_DIAG_NETWORK_STATISTICS を指定することで、接続のネットワーク統計を検索できます。

指定できる値は以下のとおりです。

SQL_NETWORK_STATISTICS_OFF (デフォルト)
接続のネットワーク統計の収集を無効にします。

SQL_NETWORK_STATISTICS_ON
接続のネットワーク統計の収集を有効にします。

SQL_NETWORK_STATISTICS_ON_SKIP_NOSERVER
接続に対してネットワーク統計の収集を使用可能にすることに加えて、明示的な COMMIT ステートメントや ROLLBACK ステートメントなど、サーバー時間が報告されていないと認識されているネットワーク・フローは省略されます。

サーバー時間が報告されていない要求は、サーバー時間をネットワーク時間から減算する計算が行われる場合、戻り情報の有用性に影響を与えることがあります。

SQL_NETWORK_STATISTICS_ON_SKIP_NOSERVER オプションは、これらの要求を報告される値から除外します。非チェーンの明示的要求のみ除外されます。自動コミットおよびチェーニングされた COMMIT ステートメントはスキップされません。

バージョン 9.7 フィックスパック 5 以降、 CLI は COMMIT および ROLLBACK で報告されるサーバー時間の統計を収集します。DB2 サーバーは、COMMIT および ROLLBACK に対するサーバー時間の報告をサポートするレベルである必要があります。

SQL_ATTR_ODBC_CURSORS
この接続属性は ODBC で定義されていますが、CLI ではサポートされません。この属性を設定または取得しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_OVERRIDE_CODEPAGE
バージョン 9.7 フィックスパック 5 以降で使用可能なこの接続属性を使用すると、CLI アプリケーションは、コード・ページ変換を行わなくても、CHARACTER データ・タイプまたは GRAPHIC データ・タイプのデータをフェッチまたは挿入できます。可能な値は次のとおりです。

- SQL_OVERRIDE_CODEPAGE_ON: CLI は、文字データまたはグラフィック・データのバインディングに対して、コード・ページ変換を実行しません。
- **SQL_OVERRIDE_CODEPAGE_OFF** (デフォルト): CLI は文字データまたはグラフィック・データのバインディングに対して、コード・ページ変換を実行します。

この属性を SQL_OVERRIDE_CODEPAGE_ON に設定する場合、データが適切なコード・ページであることを確認する必要があります。

SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE を設定した後で SQL_ATTR_OVERRIDE_CODEPAGE を SQL_OVERRIDE_CODEPAGE_ON に設定すると、CLI は CLI0126E エラー・メッセージを返します。

SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE
バージョン 9.7 フィックスパック 3 以降で使用可能なこの接続属性を使用すると、CLI アプリケーションは、データベース・コード・ページを指定できます。コード・ページは、クライアント側で使用可能になっている必要はありません。

データベース・コード・ページと同じコード・ページを指定すると、アプリケーションは、コード・ページ変換を行わなくても、CHARACTER データ・タイプのデータをフェッチまたは挿入できます。

ステートメント・ハンドルの割り振り後にこの属性を設定すると、エラー CLI0126E (無効な操作) になります。この属性をデータベースでサポートされていない値に設定すると、エラー CLI0210E (不整合なコード・ページ値のエラー) になります。

SQL_ATTR_OVERRIDE_CODEPAGE を SQL_OVERRIDE_CODEPAGE_ON に設定した後で SQL_ATTR_OVERRIDE_CHARACTER_CODEPAGE を設定すると、CLI は CLI0126E エラー・メッセージを返します。

バインド・アウト操作の際には、バインド・アウト操作中に取得したデータを保持するのに十分な大きさのバッファーを、CLI アプリケーションによって割り振るようにしてください。スペースが不十分な場合は、エラー CLI0002W が戻されます。

制約事項: この属性は、DB2 for z/OS データ・サーバーでのみサポートされます。この属性値を他のデータ・サーバーに設定しようとすると、エラー CLI0150E (ドライバーが機能しない) が戻されます。

SQL_ATTR_PACKET_SIZE

この接続属性は ODBC で定義されていますが、CLI ではサポートされません。この属性を設定または取得しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_PARC_BATCH

配列入力を使用するアプリケーションがバルク挿入、削除、または更新を完了するために、この 32 ビットの符号なし整数の接続属性は、アプリケーションが、各パラメーター・セットによって影響を受けた表の行数、またはパラメーター・セット全体で影響を受けた行の累積の数を受け取るかどうかを示します。この接続属性は、非アトミック操作に対して、DB2 バージョン 9.7 フィックスパック 5 以降で使用可能です。可能な値は次のとおりです。

- SQL_PARC_BATCH_ENABLE: CLI は、各パラメーター・セットによって影響を受けた表の行数を返します。
- **SQL_PARC_BATCH_DISABLE (デフォルト)**: CLI は、パラメーター・セット全体に対して影響を受けた行の合計数を返します。

この接続属性を SQL_PARCH_BATCH_ENABLE に設定する場合、配列を *RowCountPtr* パラメーターとして SQLRowCount () 関数で指定し、SQL_ATTR_PARAMOPT_ATOMIC を SQL_ATOMIC_NO に設定する必要があります。SQL_ATTR_PARAMOPT_ATOMIC が SQL_ATOMIC_YES に設定されていると、SQLExecute() 関数を呼び出した時点で、CLI0150E エラー・メッセージが返されます。

SQL_ATTR_PING_DB

ping 時間をマイクロ秒で取得するために `SQLGetConnectAttr()` とともに使用される 32 ビット整数。

接続が以前に確立され、データベースによって除去されている場合、値 0 が報告されます。接続がアプリケーションによってクローズされている場合、SQLSTATE の 08003 が報告されます。この接続属性は、`SQLGetConnectAttr()` によって戻せますが、 `SQLSetConnectAttr()` で設定することはできません。この属性を設定しようとすると、 SQLSTATE は 7HYC00 (ドライバーが機能しない) になります。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_PING_NTIMES

`SQLGetConnectAttr()` とともに使用される 32 ビット整数であり、アプリケーションが SQL_ATTR_PING_DB を使用する場合に CLI が実行する ping の反復回数を設定します。SQL_ATTR_PING_NTIMES を 1 より大きい値に

設定すると、SQL_ATTR_PING_DB は、CLI がデータベースへの ping を一定の回数繰り返すのに要した平均時間を返します。

この属性の有効範囲は 1 から 32767 (両端を含む) です。
SQLGetConnectAttr() は値を検査し、値がこの範囲に収まらない場合には該当するエラー・コードを返します。

SQL_ATTR_PING_REQUEST_PACKET_SIZE

SQLGetConnectAttr() とともに使用される 32 ビット整数であり、アプリケーションが SQL_ATTR_PING_DB を使用する場合に CLI が使用する ping パケットのサイズを設定します。

この属性の有効範囲は 1 から 32767 (両端を含む) です。
SQLGetConnectAttr() は値を検査し、値がこの範囲に収まらない場合には該当するエラー・コードを返します。

SQL_ATTR_QUIET_MODE

32 ビット・プラットフォーム特定のウィンドウ・ハンドル。

アプリケーションがこのオプションを指定して SQLSetConnectAttr() を呼び出したことがない場合、 CLI はこのオプションに、
SQLGetConnectAttr() に対する NULL の親ウィンドウ・ハンドルを戻し、NULL の親ウィンドウ・ハンドルを使用してダイアログ・ボックスを表示します。例えば、エンド・ユーザーが (CLI 初期設定ファイルの項目を介して) オプティマイザー情報を表示するよう求めると、 CLI は NULL のウィンドウ・ハンドルを使用してこの情報が入っているダイアログ・ボックスを表示します。 (プラットフォームによっては、ダイアログ・ボックスが画面の中央に表示されます。)

ValuePtr を NULL に設定すると、CLI はダイアログ・ボックスを表示しません。エンド・ユーザーがオプティマイザー見積もりを表示するよう求める場合には、アプリケーションが明示的にそのようなダイアログ・ボックスをすべて抑止するので、 CLI はこれらの見積もりを表示しません。

ValuePtr が NULL ではない場合、アプリケーションの親ウィンドウ・ハンドルになるはずです。 CLI は、このハンドルを使ってダイアログ・ボックスを表示します。 (プラットフォームによっては、ダイアログ・ボックスがアプリケーションのアクティブ・ウィンドウとの関係において、中央に表示されます。)

注: この接続オプションを、 SQLDriverConnect() ダイアログ・ボックスを隠すために使用することはできません。 (隠すためには、*fDriverCompletion* 引数を SQL_DRIVER_NOPROMPT に設定します。)

SQL_ATTR_RECEIVE_TIMEOUT

試行を終了し、通信タイムアウト・エラーを生成する前の、確立された接続でのサーバーからの応答をクライアントが待機する秒数を指定する、32 ビット整数値。デフォルト値の 0 は、クライアントが応答を無制限に待機することを示します。受信タイムアウトは、接続の確立中は効果がありません。また、TCP/IP プロトコルに対してのみサポートされ、その他のプロトコルの場合は無視されます。サポートされる値は、 0 から 32767 までの整数です。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_REOPT

特殊レジスターまたはパラメーター・マーカーを含む SQL ステートメントに対して照会最適化を有効にする 32 ビット整数値。コンパイラーによって選択されるデフォルトの推定値の代わりに、特殊レジスターまたはパラメーター・マーカーに対して照会実行時に使用できる値を使用することによって、最適化が生じます。属性の有効値は、次のとおりです。

- **2 = SQL_REOPT_NONE (デフォルト):** 照会の実行時に照会最適化は行われません。コンパイラーによって選択されるデフォルトの推定値が、特殊レジスターまたはパラメーター・マーカーに対して使用されます。デフォルトの NULLID パッケージ・セットは、動的 SQL ステートメントの実行に使用されます。
- 3 = SQL_REOPT_ONCE: 照会最適化は、照会実行時に一度、初めて照会を実行するときに生じます。NULLIDR1 パッケージ・セットが使用されますが、これは REOPT ONCE バインド・オプションとバインドされています。
- 4 = SQL_REOPT_ALWAYS: 照会最適化または再最適化は、照会の実行時に、毎回必ず生じます。NULLIDRA パッケージ・セットが使用されますが、これは REOPT ALWAYS バインド・オプションとバインドされています。

NULLIDR1 および NULLIDRA は予約済みパッケージ・セット名で、使用時には REOPT ONCE および REOPT ALWAYS が暗黙指定されます。これらのパッケージ・セットは、これらのコマンドで明示的に作成する必要があります。

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

SQL_ATTR_REOPT と SQL_ATTR_CURRENT_PACKAGE_SET を同時に指定することはできません。したがって、一方を設定すると、他方は許可されません。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB

32 ビット整数値。 OLE DB クライアント・カーソル・エンジンおよび OLE DB .NET Data Provider CommandBuilder オブジェクトは、IBM DB2 OLE DB Provider によって提供される列情報に基づいて、UPDATE および DELETE ステートメントを生成します。生成されたステートメントの WHERE 節に LONG タイプが含まれる場合には、ステートメントが失敗します。なぜなら、等価演算子を使った検索で LONG タイプは使用できないからです。可能な値は次のとおりです。

- **0 (デフォルト)**: LONG タイプ (LONG VARCHAR、LONG VARCHAR FOR BIT DATA、LONG VARGRAPHIC、および LONG VARGRAPHIC FOR BIT DATA) は、DBCOLUMNFLAGS_ISLONG フラグを設定しません。そのため、WHERE 節でその列が使用されることがあります。

- 1: IBM DB2 OLE DB Provider は LONG タイプ (LONG VARCHAR、LONG VARCHAR FOR BIT DATA、LONG VARGRAPHIC、および LONG VARGRAPHIC FOR BIT DATA) を、DBCOLUMNFLAGS_ISLONG フラグを設定して報告します。これは、WHERE 節で long 列が使用されるのを防ぎます。

この属性は、以下のデータベース・サーバーによってサポートされます。

- DB2 for z/OS
  - バージョン 6 (PTF UQ93891 適用済み)
  - バージョン 7 (PTF UQ93889 適用済み)
  - バージョン 8 (PTF UQ93890 適用済み)
  - バージョン 8 より後のバージョン (PTF は不要)
- DB2 Database for Linux, UNIX, and Windows
  - バージョン 8.2 (バージョン 8.1、フィックスパック 7 と等価) 以降

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_REPORT_SEAMLESSFAILOVER_WARNING

シームレス・フェイルオーバーが要求中に発生した場合に、実行要求で警告メッセージを返すかどうかを指定する 32 ビットの符号なしの整数値。この接続属性は、DB2 バージョン 9.7 フィックスパック 5 以降で使用可能です。指定可能な値は次のとおりです。

- SQL_REPORT_SEAMLESSFAILOVER_WARNING_YES: シームレス・フェイルオーバーが実行要求中に発生した場合、警告メッセージが返されます。
- **SQL_REPORT_SEAMLESSFAILOVER_WARNING_NO (デフォルト)**: シームレス・フェイルオーバーが実行要求中に発生した場合、警告メッセージが返されません。

SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN

日時オーバーフローがエラー (SQLSTATE 22008) または警告 (SQLSTATE 01S07) のどちらになるかを指定する 32 ビットの符号なし整数値。可能な値は次のとおりです。

- **0 (デフォルト)**: 日時オーバーフローはエラーにつながる (SQLSTATE 22008)。
- 1: 日時オーバーフローは警告につながる (SQLSTATE 01S07)。

SQL_ATTR_RETRYONERROR

CLI は、アプリケーション・パラメーターの不正なバインドなどの致命的ではないエラーからのリカバリーを、失敗した SQL ステートメントに関する追加情報を検索してからそのステートメントを再び実行することによって行います。検索される追加情報には、データベース・カタログ表からの入力パラメーター情報が含まれます。 CLI がエラーから正常にリカバリーできる場合は、デフォルトでは、アプリケーションへのエラー報告はされません。

CLI/ODBC 構成キーワード ReportRetryErrorsAsWarnings を使用することにより、エラー・リカバリー警告がアプリケーションに戻されるようにするかどうかを設定できます。

注: いったん CLI が正常にエラー・リカバリーを完了すると、アプリケーションが通常とは異なる動作をする場合がありますが、それは CLI が元の SQLBindParameter() 関数呼び出しで提供された情報ではなく、それに続けて実行した SQL ステートメントに対するリカバリー中に収集されたカタログ情報を使用するためです。このような動作が望ましくない場合は、RetryOnError を 0 に設定し、強制的に CLI がリカバリーを試行しないようにします。しかし、ステートメント・パラメーターを正しくバインドするように、アプリケーションを変更する必要があります。

SQL_ATTR_SERVER_MSGTXT_MASK

CLI がサーバーからエラー・メッセージをいつ要求するかを示すために使用される、32 ビット整数値。この属性は、SQL_ATTR_SERVER_MSGTXT_SP 属性と組み合わせて使用されます。属性は以下の値に設定できます。

- **SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST (デフォルト)**: CLI は、ローカル・メッセージ・ファイルを最初に調べて、メッセージを検索できるかどうかを確かめます。一致する SQLCODE が見つからない場合、CLI はサーバーから情報を要求します。
- SQL_ATTR_SERVER_MSGTXT_MASK_WARNINGS: CLI は常に、警告に関するサーバーからのメッセージ情報を要求しますが、エラー・メッセージがローカル・メッセージ・ファイルから取り出されます。
- SQL_ATTR_SERVER_MSGTXT_MASK_ERRORS: CLI は常に、エラーに関するサーバーからのメッセージ情報を要求しますが、警告メッセージがローカル・メッセージ・ファイルから取り出されます。
- SQL_ATTR_SERVER_MSGTXT_MASK_ALL: CLI は常に、エラーと警告メッセージの両方に関する情報をサーバーから要求します。

445 ページの『ServerMsgMask CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_SERVER_MSGTXT_SP

SQLCA を基にしてエラー・メッセージを生成するために使用されるストアード・プロシージャーを識別するための文字ストリングを指すポインター。これは、DB2 for z/OS などのサーバーからエラー情報を取り出す際に役立ちます。属性は以下の値に設定できます。

- SYSIBM.SQLCAMESSAGE: メッセージ・テキストを DB2 for z/OS サーバーから取り出すために呼び出されるデフォルト・プロシージャー。この属性を設定しない場合は、このプロシージャーが呼び出されます。
- DSNACCMG: メッセージ・テキストを DB2 for z/OS バージョン 7 サーバーから取り出すために呼び出されるデフォルト・プロシージャー。メッセージ・テキストを DB2 for z/OS バージョン 8 以降から取り出すためには、SYSIBM.SQLCAMESSAGE プロシージャーが呼び出されます。DSNACCMG は、DB2 for z/OS バージョン 9 では推奨されておらず、将来のリリースで除去される可能性があります。

- ユーザーが作成したストアード・プロシージャー

この属性を使用するアプリケーションは、SQL_ATTR_SERVER_MSGTXT_MASK 属性も設定して、メッセージ情報をサーバーから取り出すために CLI がいつこのプロシージャーを呼び出すかを示すことができます。 SQL_ATTR_SERVER_MSGTXT_MASK が設定されていない場合、デフォルトで、ローカル・メッセージ・ファイルを最初に調べます (SQL_ATTR_SERVER_MSGTXT_MASK の SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST を参照してください)。

475 ページの『UseServerMsgSP CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_SESSION_TIME_ZONE
サーバー・セッション・タイム・ゾーン情報が含まれる、±hh:mm 形式のヌル終了文字ストリング。これは、設定のみが可能な属性です。サポートされるタイム・ゾーン値は、-12:59 から +14:00 までの範囲です。

SQL_ATTR_SQLCODEMAP
SQLCODE マッピングをデフォルトに設定するか、オフにするかを指定します。可能な値は次のとおりです。

- **MAP (デフォルト)**: SQLCODE マッピングが設定されます。
- NOMAP: SQLCODE マッピングがオフになります。

SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB
32 ビット整数値。 Microsoft OLE DB 仕様は、IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) が列 TABLE_CATALOG、TABLE_SCHEMA、TABLE_NAME、COLUMN_NAME によってソートされた行セットを戻すことを要求します。 IBM DB2 OLE DB Provider はこの仕様に準拠していますが、Microsoft ODBC Bridge provider (MSDASQL) を使用するアプリケーションは、一般に、行セットが ORDINAL_POSITION によってソートされるようにコード化されています。可能な値は次のとおりです。

- **0 (デフォルト)**: IBM DB2 OLE DB Provider は、TABLE_CATALOG、TABLE_SCHEMA、TABLE_NAME, COLUMN_NAME 列によってソートされた行セットを戻します。
- 1: IBM DB2 OLE DB Provider は、ORDINAL_POSITION によってソートされた行セットを戻します。

この属性は、以下のデータベース・サーバーによってサポートされます。

- DB2 for z/OS
  - バージョン 6 (PTF UQ93891 適用済み)
  - バージョン 7 (PTF UQ93889 適用済み)
  - バージョン 8 (PTF UQ93890 適用済み)
  - バージョン 8 より後のバージョン (PTF は不要)
- DB2 Database for Linux, UNIX, and Windows
  - バージョン 8.2 (バージョン 8.1、フィックスパック 7 と等価) 以降

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_STMT_CONCENTRATOR
リテラル値を含む動的ステートメントがステートメント・キャッシュを使用するかどうかを指定します。

- SQL_STMT_CONCENTRATOR_OFF - ステートメント・コンセントレーターの動作は無効です。
- SQL_STMT_CONCENTRATOR_WITH_LITERALS - サーバーがサポートしている状況の場合、ステートメント・コンセントレーターが有効になり、リテラル値を含む動的ステートメントはステートメント・キャッシュを使用します。 例えば、ステートメントにパラメーター・マーカー、名前付きパラメーター・マーカー、またはリテラルとパラメーター・マーカーおよび名前付きパラメーター・マーカーの組み合わせが含まれている場合、ステートメント・コンセントレーターは有効になりません。

451 ページの『StmtConcentrator CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_STREAM_GETDATA
`SQLGetData()` 関数のデータ出力ストリームが最適化されるかどうかを示す 32 ビット符号なし整数。値は以下のとおりです。

- **0 (デフォルト)**: CLI はクライアント上のすべてのデータをバッファーに入れます。
- 1: データをバッファーに入れる必要のないアプリケーションで Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするサーバー上のデータを照会している場合は、1 を指定することによって、データをバッファーに入れる必要がないことを示してください。 CLI クライアントは、データ出力ストリームを最適化します。

Dynamic Data Format がサーバーによってサポートされない場合、このキーワードは無視されます。

StreamGetData が 1 に設定されている場合、出力バッファーに入れて戻すためにまだ使用できるバイト数を CLI が判別できない場合、切り捨て発生時に `SQLGetData()` は長さとして SQL_NO_TOTAL (-4) を戻します。それ以外の場合、`SQLGetData()` は、まだ使用できるバイト数を戻します。

452 ページの『StreamGetData CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_SYNC_POINT
この属性は、DB2 UDB バージョン 8 から使用すべきでない属性となりました。

SQL_ATTR_TRACE
この接続属性は、ODBC Driver Manager 用のアプリケーションで設定できます。 CLI Driver にこの接続属性を設定しようとすると、SQLSTATE は HYC00 (ドライバーが機能しない) になります。

この接続属性を使用する代わりに、 457 ページの『Trace CLI/ODBC 構成キーワード』を使用して CLI トレース機能を設定することができます。あるいは、環境属性 SQL_ATTR_TRACE を使用して、トレース・フィーチャーを構成することもできます。環境属性は ODBC Driver Manager の接続属性と同じ構文を使用しないことに注意してください。

SQL_ATTR_TRACEFILE

この接続属性は ODBC で定義されていますが、CLI ではサポートされません。この属性を設定または取得しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

この属性を使用する代わりに 463 ページの『TraceFileName CLI/ODBC 構成キーワード』を使用して、CLI トレース・ファイル名を設定します。

SQL_ATTR_TRANSLATE_LIB

この接続属性は ODBC で定義されていますが、CLI ではサポートされません。他のプラットフォームでこの属性を設定または取得しようとすると、SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_TRANSLATE_OPTION

この接続属性は ODBC で定義されていますが、CLI ではサポートされません。他のプラットフォームでこの属性を設定または取得しようとすると、SQLSTATE は HYC00 (ドライバーが機能しない) になります。

SQL_ATTR_TRUSTED_CONTEXT_PASSWORD

パスワードを含む、ユーザー定義ストリング。トラステッド接続上のユーザーを切り替えるときにデータベース・サーバーがパスワードを必要とする場合は、この属性を使用します。この属性は、属性 SQL_ATTR_TRUSTED_CONTEXT_USERID を設定した後、およびデータベース・サーバーにアクセスする SQL ステートメントを実行する前に設定してください。この属性を設定する前に SQL_ATTR_TRUSTED_CONTEXT_USERID が設定されていない場合、エラー (CLI0198E) が戻されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_TRUSTED_CONTEXT_USERID

ユーザー ID を含む、ユーザー定義ストリング。ユーザーを切り替えるには、既存のトラステッド接続上でこれを使用します。トラステッド接続を作成する際には、これを使用しないでください。

この属性を設定した後、次にデータベース・サーバーにアクセスする SQL ステートメントを実行すると、ユーザーの切り替えが行われます。(SQLSetConnectAttr はデータベース・サーバーにアクセスしません。) ユーザーの切り替えが成功する場合、この属性内のユーザー ID が接続の新規ユーザーになります。ユーザーの切り替えが失敗すると、切り替えを開始した呼び出しは、失敗の理由を示すエラーを戻します。

ID サーバーを使用していない場合、ユーザー ID はデータベース・サーバー上の有効な許可 ID でなければなりません。その場合、ID サーバーが認識する任意のユーザー名を使用できます。 (ID サーバーを使用している場合は、SQL_ATTR_USER_REGISTRY_NAME も参照してください。)

接続ハンドルがまだデータベースに接続されていない間にこの属性を設定する場合、または接続がトラステッド接続ではない場合、エラー (CLI0197E) が戻されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_TXN_ISOLATION

ConnectionHandle で参照され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 ビットのビット・マスク。 *ValuePtr* の有効値は、実行時に *fInfoType* を SQL_TXN_ISOLATION_OPTIONS に設定して `SQLGetInfo()` を呼び出せば判別できます。次の値は CLI では受け入れられますが、各サーバーではこれらの分離レベルのうちの 1 つのサブセットしかサポートできないことがあります。

- SQL_TXN_READ_UNCOMMITTED - ダーティー読み取り、反復不能読み取り、幻像読み取りが可能です。
- **SQL_TXN_READ_COMMITTED (デフォルト)** - ダーティー読み取りが不可能です。反復不能読み取りと幻像読み取りが可能です。
- SQL_TXN_REPEATABLE_READ - ダーティー読み取りと反復不能読み取りが不可能です。幻像読み取りが可能です。
- SQL_TXN_SERIALIZABLE - トランザクションがシリアライズ可能です。ダーティー読み取り、反復不能読み取り、幻像読み取りが不可能です。
- SQL_TXN_NOCOMMIT - 操作が正常に終了したときに、変更内容が有効にコミットされます。明示コミットやロールバックはできません。これは、自動コミットに似ています。これは SQL92 分離レベルではありませんが、DB2 UDB for AS/400 のみがサポートする IBM 定義の拡張機能です。

IBM の用語では、
- SQL_TXN_READ_UNCOMMITTED は、非コミット読み取りです。
- SQL_TXN_READ_COMMITTED は、カーソル固定です。
- SQL_TXN_REPEATABLE_READ は、読み取り固定です。
- SQL_TXN_SERIALIZABLE は、反復可能読み取りです。

このオプションは、いずれかのステートメント・ハンドル上にオープン・カーソルがある場合や、この接続に未解決のトランザクションがある場合は指定できません。それ以外の場合は、関数呼び出し時に SQL_ERROR (SQLSTATE **S1**011) が戻されます。

この属性 (または対応するキーワード) を使用できるのは、デフォルトの分離レベルが使用される場合だけです。アプリケーションが特定の分離レベルを設定する場合は、この属性を設定しても効果はありません。

注: ステートメント・ハンドルでトランザクション分離レベルの設定を許可する IBM 拡張機能があります。詳細は、SQL_ATTR_STMTTXN_ISOLATION ステートメント属性の項を参照してください。

SQL_ATTR_USE_TRUSTED_CONTEXT

信頼できるコンテキストをサポートする DB2 データベース・サーバーに接

続する際に、作成先の接続をトラステッド接続にする場合には、この属性を設定します。この属性が SQL_TRUE に設定されており、データベース・サーバーが接続を信頼できると判断する場合、その接続はトラステッド接続です。この属性が設定されていない場合、この属性が SQL_FALSE に設定されている場合、データベース・サーバーが信頼できるコンテキストをサポートしない場合、またはデータベース・サーバーが接続を信頼できないと判断する場合、通常の接続が代わりに作成され、警告 (SQLSTATE 01679) が戻されます。この値は、接続が初めて確立される前か、または SQLDisconnect() 関数の呼び出しに続いて確立される前にのみ指定できます。

SQL_ATTR_USER_REGISTRY_NAME

この属性は、サーバー上で ID マッピング・サービスを使用するユーザーを認証する際にのみ使用されます。これは、ID マッピング・レジストリーに名前を付けるユーザー定義ストリングに設定されます。レジストリー名のフォーマットは、使用される ID マッピング・サービスに応じて変化します。この属性を指定することにより、提供したユーザー名がこのレジストリーにあることがサーバーに通知されます。

この属性を設定した後、次に通常接続を確立しようとするとき、次にトラステッド接続を確立しようとするとき、または次にトラステッド接続でユーザー ID を切り替えようとするときに、値が使用されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_WCHARTYPE

アプリケーションで使用する wchar_t (SQLDBCHAR) 文字フォーマットを、2 バイトの環境で指定する 32 ビット整数。このオプションには、wchar_t データをマルチバイト・フォーマットまたはワイド文字フォーマットのどちらにするかの選択の融通性があります。このオプションに使用できる 2 つの値は、次のとおりです。

- SQL_WCHARTYPE_CONVERT: データベースにある GRAPHIC SQL データとアプリケーション変数間で、文字コードが変換されます。この変換により、アプリケーションで広幅文字ストリング (例えば、L リテラル、'wc' ストリング関数) を処理する ANSI C メカニズムを最大限に活用できるようになります。データベースと通信する前に、データをマルチバイト・フォーマットに明示的に変換する必要はありません。不利な点としては、暗黙的な変換により、アプリケーションの実行時パフォーマンスが影響を受け、メモリー所要量が増える可能性があります。 WCHARTYPE CONVERT を行う必要がある場合、コンパイル時に C プリプロセッサー・マクロ SQL_WCHART_CONVERT を定義してください。これにより、DB2 ヘッダー・ファイルにある特定の定義で、データ・タイプ sqldbchar の代わりに wchar_t を使用することになります。
- **SQL_WCHARTYPE_NOCONVERT (デフォルト)**: アプリケーションとデータベースとの間で、暗黙的な文字コード変換は行われません。アプリケーション変数のデータは、非代替 DBCS 文字としてデータベースへ送信され、かつデータベースから受信します。この送受信の場合、アプリケーションのパフォーマンスは向上しますが、アプリケーションが wchar_t (SQLDBCHAR) アプリケーション変数で広幅文字データを使用しなくな

ったり、 wcstombs() および mbstowcs() ANSI C 関数を明示的に呼び出して、データをデータベースとやりとりするときに、そのデータをマルチバイト・フォーマットに変換したり、このフォーマットから変換したりする、という不利な点があります。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_XML_DECLARATION
XML データが暗黙的にシリアライズされるときに XML 宣言のエレメントが XML データに追加されることを示す 32 ビット符号なし整数。この属性は XMLSERIALIZE 関数の結果に影響を与えません。この属性を、必要な各コンポーネントの合計に設定します。

- 0: 出力バッファーに追加される宣言またはバイト・オーダー・マーク (BOM) はありません。
- 1: ターゲット・エンコードが UTF-16 または UTF-32 の場合、該当するエンディアン (リトル・エンディアンまたはビッグ・エンディアン) のバイト・オーダー・マーク (BOM) が出力バッファーの前に付加されます。(UTF-8 BOM がある場合でも、ターゲット・エンコードが UTF-8 であっても、DB2 は UTF-8 BOM を生成しません。)
- 2: XML バージョンだけを含む最小の XML 宣言が生成されます。
- 4: ターゲット・エンコードを識別するエンコード属性が、生成された XML 宣言に追加されます。そのため、この設定が有効になるのは、この属性の値を計算する際に設定 2 も含まれている場合に限ります。

`SQLSetConnectAttr()` または `SQLSetConnectOption()` を使用してその他の値を設定しようとすると、CLI0191E (SQLSTATE HY024) エラーが出され、値は未変更のままになります。

デフォルト設定は 7 です。これは、XML バージョンとエンコード属性を含む BOM および XML 宣言が暗黙的なシリアライズ中に生成されることを示します。

この設定は、値の変更後に割り振られるステートメント・ハンドルに影響を与えます。既存のステートメント・ハンドルは元の値を保持します。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

ステートメント属性 (CLI) のリスト

現在定義されている属性とそれらが導入された CLI または ODBC のバージョンは、このセクションで記載されているとおりです。さまざまなデータ・ソースを利用できるよう、さらに多くの属性が今後定義される見込みです。

SQL_ATTR_ALLOW_INTERLEAVED_GETDATA
アプリケーションが、Dynamic Data Format をサポートするデータ・サーバーを照会するときに、以前にアクセスした LOB 列に対して `SQLGetData()` を呼び出し、 `SQLGetData()` への以前の呼び出し時のデータ・オフセット位置を維持できるかどうかを指定します。この属性の値は次のうちの 1 つとなります。

- SQL_ALLOW_INTERLEAVED_GETDATA_OFF - デフォルト設定は、アプリケーションが以前にアクセスした LOB 列に対して SQLGetData() を呼び出すことを許可しません。
- SQL_ALLOW_INTERLEAVED_GETDATA_ON - このキーワードは、Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするデータベース・サーバーへの接続にのみ影響を与えます。このオプションを指定すると、アプリケーションは以前にアクセスした LOB 列に対して SQLGetData() を呼び出し、以前の読み取り時にアプリケーションが読み取りを停止したところから LOB データの読み取りを開始することができます。

364 ページの『AllowInterleavedGetData CLI/ODBC 構成キーワード』 の設定は、この動作を接続レベルで指定する別の方法です。

SQL_ATTR_ALLOW_INTERLEAVED_GETDATA 接続属性は、IDS データ・サーバーではサポートされません。

SQL_ATTR_APP_PARAM_DESC
ステートメント・ハンドルで後続の SQLExecute() および SQLExecDirect() 呼び出しを行うための APD へのハンドル。この属性の初期値は、ステートメントの初期割り当て時に暗黙的に割り当てられる記述子です。この属性が SQL_NULL_DESC に設定されていると、明示的に割り当てられた APD ハンドルは、今まで関連付けられていたステートメント・ハンドルとの関連付けを断たれ、ステートメント・ハンドルは、暗黙的に割り当てられる APD ハンドルに戻ります。

この属性は、別のステートメントに暗黙的に割り当てられた記述子ハンドル、または同じステートメントで暗黙的に設定された別の記述子ハンドルには設定することができません。つまり、暗黙的に割り当てられた記述子ハンドルは、 1 つのステートメントまたは 1 つの記述子ハンドルにしか関連付けできないということです。

この属性は、接続レベルでは設定できません。

SQL_ATTR_APP_ROW_DESC
ステートメント・ハンドルでの以後のフェッチを行うための ARD へのハンドル。この属性の初期値は、ステートメントの初期割り当て時に暗黙的に割り当てられる記述子です。この属性が SQL_NULL_DESC に設定されていると、明示的に割り当てられた ARD ハンドルは、今まで関連付けられていたステートメント・ハンドルとの関連付けを断たれ、ステートメント・ハンドルは、暗黙的に割り当てられる ARD ハンドルに戻ります。

この属性は、別のステートメントに暗黙的に割り当てられた記述子ハンドル、または同じステートメントで暗黙的に設定された別の記述子ハンドルには設定することができません。つまり、暗黙的に割り当てられた記述子ハンドルは、 1 つのステートメントまたは 1 つの記述子ハンドルにしか関連付けできないということです。

この属性は、接続レベルでは設定できません。

SQL_ATTR_APP_USES_LOB_LOCATOR
アプリケーションが LOB ロケーターを使用するかどうかを示す 32 ビット符号なし整数。この属性の値は次のうちの 1 つとなります。

- **1 (デフォルト)**: アプリケーションが LOB ロケーターを使用するかどうかを示します。
- 0: LOB ロケーターを使用しないアプリケーションで Dynamic Data Format (プログレッシブ・ストリーミングとも呼ばれる) をサポートするサーバー上のデータを照会している場合は、0 を指定することによって、LOB ロケーターが使用されないことを示し、LOB データの戻りが最適化されるようにしてください。

ストアード・プロシージャーの結果セットの場合、このキーワードは無視されます。

キーワードが 0 に設定されている場合、アプリケーションが SQLBindCol() を使用して LOB ロケーターを結果セットにバインドすると、SQLFetch() 関数から無効な変換エラーが戻されます。

366 ページの『AppUsesLOBLocator CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_ASYNC_ENABLE

指定したステートメントで呼び出される関数が非同期で実行されるかどうかを指定する 32 ビット整数値。

- **SQL_ASYNC_ENABLE_OFF** = Off (デフォルト値)
- SQL_ASYNC_ENABLE_ON = On

関数が非同期に呼び出されると、元の関数が SQL_STILL_EXECUTING 以外のコードを戻すまでは、元の関数、 SQLAllocHandle()、SQLCancel()、SQLSetStmtAttr()、SQLGetDiagField()、SQLGetDiagRec()、または SQLGetFunctions() だけが、ステートメント・ハンドルで呼び出せます。同じ接続下にある他のステートメント・ハンドルで他の関数が呼び出されると、SQL_ERROR が SQLSTATE HY010 (関数シーケンス・エラー) を伴って戻されます。

CLI はステートメント・レベルの非同期実行をサポートするため、ステートメント属性 SQL_ATTR_ASYNC_ENABLE を設定できます。その初期値は、ステートメント・ハンドルが割り振られたときの同じ名前を持つ接続レベル属性の値と同じです。

以下の関数は、非同期に実行できます。SQLBulkOperations()、SQLColAttribute()、SQLColumnPrivileges()、SQLColumns()、SQLDescribeCol()、SQLDescribeParam()、SQLExecDirect()、SQLExecute()、SQLExtendedFetch()、SQLExtendedPrepare()、SQLFetch()、SQLFetchScroll()、SQLForeignKeys()、SQLGetData()、SQLGetLength()、SQLGetPosition()、SQLMoreResults()、SQLNumResultCols()、SQLParamData()、SQLPrepare()、SQLPrimaryKeys()、SQLProcedureColumns()、SQLProcedures()、SQLRowCount()、SQLSetPos()、SQLSpecialColumns()、SQLStatistics()、SQLTablePrivileges()、SQLTables()。

注: どの Unicode 等価関数も非同期で呼び出すことができます。バージョン 9.7 フィックスパック 4 以降、SQL_ATTR_ASYNC_ENABLE 属性は SQL_ATTR_USE_LOAD_API と一緒に使用することができます。

SQL_ATTR_BLOCK_FOR_NROWS
結果セットのフェッチ時に、サーバーから戻される規定のブロック・サイズを何行にするかを指定する 32 ビット整数。1 つ以上のデータ・ブロックで構成される読み取り専用の大きな結果セットの場合、ブロック・サイズを大きい値に指定することによって、クライアントからのサーバー・ブロックの同時要求数が小さくなり、パフォーマンスが向上する場合があります。デフォルト値は 0 であり、その場合はデフォルトのブロック・サイズがサーバーから戻されます。

SQL_ATTR_BLOCK_LOBS
LOB データ・タイプを戻す結果セットのブロックを有効にするかどうかを指定する Boolean 属性。この属性は、デフォルトでは 0 (false) に設定されていますが、1 (true) に設定した場合は、 LOB データ・タイプを戻す結果セットのブロックをサポートするサーバーにアクセスする際に、 1 回のフェッチ要求に対し、単一の照会ブロックに完全に収まる行に対応する LOB データがすべて戻されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CALL_RETURN
ストアード・プロシージャー実行後に取り出されることになる読み取り専用属性。ストアード・プロシージャーが実行に失敗した場合、この属性から戻される値は -1 になります (その実行可能ストアード・プロシージャーを含むライブラリーが見つからない場合など)。ストアード・プロシージャーは正常に実行されたが、戻りコードが負の数の場合 (例えば、表にデータを挿入する際にデータの切り捨てが実行された場合)、
SQL_ATTR_CALL_RETURN は、そのストアード・プロシージャーの実行時に SQLCA の sqlerrd(1) フィールドに設定された値を戻します。

SQL_ATTR_CHAINING_BEGIN
準備済みの 1 つのステートメントに関する複数の `SQLExecute()` 要求をサーバーに送信する前に、 DB2 がそれらをまとめてチェーニングすることを指定する 32 ビット整数。このフィーチャーは「CLI 配列入力チェーニング」と呼ばれます。準備済みステートメントに関連付けられたすべての `SQLExecute()` 要求は、SQL_ATTR_CHAINING_END ステートメント属性が設定されるまで、または使用できるバッファー・スペースがチェーニングされた行によって消費されるまで、サーバーには送られません。このバッファーのサイズは、ローカル・クライアント・アプリケーションの場合は **aslheapsz** データベース・マネージャー構成パラメーターによって、またクライアント/サーバー構成の場合は **rqrioblk** データベース・マネージャー構成パラメーターによって定義されます。

この属性を CLI/ODBC 構成キーワード **ArrayInputChain** と共に使用すると、配列サイズを指定することなく配列入力を実行できます。詳しくは、**ArrayInputChain** の説明を参照してください。

注: この属性に設定される特定の 32 ビット整数値には、 CLI にとって特に意味はありません。この属性を何らかの 32 ビット整数値に設定することで、 CLI 配列入力チェーニング・フィーチャーが有効になります。

SQL_ATTR_CHAINING_END

以前に SQL_ATTR_CHAINING_BEGIN ステートメント属性を設定することによって有効になった CLI 配列入力データの動作を終了することを指定する 32 ビット整数。 SQL_ATTR_CHAINING_END を設定すると、チェーニングされたすべての SQLExecute() 要求がサーバーに送信されます。この属性が設定されたなら、その後、SQLRowCount() を呼び出すことによって、SQL_ATTR_CHAINING_BEGIN と SQL_ATTR_CHAINING_END のペアの間にチェーニングされたすべての SQLExecute() ステートメントの合計行カウントを調べることができます。チェーニングされたステートメントのエラー診断情報は、 SQL_ATTR_CHAINING_END 属性の設定後に使用できるようになります。

この属性を CLI 構成キーワード **ArrayInputChain** と共に使用すると、配列サイズを指定することなく配列入力を実行できます。詳しくは、**ArrayInputChain** の説明を参照してください。

注: この属性に設定される特定の 32 ビット整数値には、 CLI にとって特に意味はありません。この属性を何らかの 32 ビット整数値に設定することで、 SQL_ATTR_CHAINING_BEGIN 設定時に有効だった CLI 配列入力チェーニング・フィーチャーが無効になります。

SQL_ATTR_CLIENT_LOB_BUFFERING

バインドされていない LOB 列について、 LOB ロケーターまたはその基になる LOB データが結果セットに入れて戻されるかどうかを指定します。デフォルトでは、ロケーターが戻されます。バインドされていない LOB をアプリケーションがフェッチしてから、その基になる LOB データを取り出すことが通常必要になるのであれば、初めから LOB データを取り出すようにすることによって、アプリケーションのパフォーマンスが改善されることがあります。このアクションによって、同期の待機とネットワーク・フローの数が少なくなります。この属性に指定可能な値は、以下のとおりです。

- SQL_CLIENTLOB_USE_LOCATORS (デフォルト) - LOB ロケーターが戻されます。
- SQL_CLIENTLOB_BUFFER_UNBOUND_LOBS - 実際の LOB データが戻されます。

SQL_ATTR_CLOSE_BEHAVIOR

カーソルがクローズする時、カーソルの操作時に獲得された読み取りロックの解放を DB2 サーバーに試行させるかどうかを指定する 32 ビット整数値。次のどちらかに設定できます。

- **SQL_CC_NO_RELEASE** - 読み取りロックは解放されません。これはデフォルトです。
- SQL_CC_RELEASE - 読み取りロックは解放されます。

分離 UR または CS でオープンされているカーソルの場合、カーソルが行から移動した後、読み取りロックは保持されません。分離 RS または RR でオープンされているカーソルの場合、 SQL_ATTR_CLOSE_BEHAVIOR はそれらの分離レベルのいくつかを変更し、 RR カーソルで非反復読み取りまたは幻像読み取りが行われることがあります。

元々 RR または RS であるカーソルが、クローズされた後で SQL_ATTR_CLOSE_BEHAVIOR で再度オープンされると、新しい読み取りロックが獲得されます。

この属性は接続レベルでも設定できますが、接続レベルで設定する場合は、この属性が設定された後にオープンされるステートメント・ハンドルのカーソルの振る舞いしか作用しません。

詳細は、SQLCloseCursor() 関数の項を参照してください。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_CLOSEOPEN

カーソルのクローズとオープンにかかる時間を短くするために、同じハンドルを使用して 2 番目のカーソルがオープンされると、 DB2 はオープンされているカーソルを自動的にクローズします。このように、クローズ要求がオープン要求とチェーニングされ、 2 つのステートメントが結合されて (2 つの要求ではなく) 1 つのネットワーク要求になると、ネットワーク・フローは少なくなります。

- **0** = DB2 は、正規の ODBC データ・ソースとして動作します。クローズ・ステートメントとオープン・ステートメントはチェーニングされず、オープン・カーソルがあるとエラーが戻されます。この動作はデフォルトです。
- 1 = クローズ・ステートメントとオープン・ステートメントをチェーニングします。

以前の CLI アプリケーションでは、カーソルが明示的にクローズされるように設計されているので、このデフォルト値は役立ちません。しかし、新しいアプリケーションでは、カーソルを明示的にクローズするのではなく、後続のオープン要求時に CLI にカーソルをクローズさせることによって、この動作を利用することができます。

SQL_ATTR_COLUMNWISE_MRI

DB2 for z/OS サーバーに接続されている CLI アプリケーションが配列入力チェーニングを INSERT 操作の列方向配列入力に変換できるようにする 32 ビットの符号なし整数。この属性は、バージョン 9.7 フィックスパック 5 以降で使用可能です。指定可能な値は次のとおりです。

- **SQL_COLUMNWISE_MRI_OFF (デフォルト)**:CLI は、チェーニング・データを列方向配列入力に変換しません。
- SQL_COLUMNWISE_MRI_ON: CLI は、配列入力チェーニングを列方向配列入力に変換します。DB2 for z/OS の複数行挿入 (MRI) フィーチャーは、データが列方向配列の形式であることを想定しています。アプリケーションが配列入力チェーニングを使用している場合、この変換によって、SQLExecute () を呼び出すたびにデータが圧縮された配列の形式で送信されるため、アプリケーションのパフォーマンスを最適化するのに役立ちます。配列入力チェーニングについて詳しくは、SQL_ATTR_CHAINING_BEGIN を参照してください。

DB2 for z/OS サーバー以外では、CLI は自動的にチェーニング・データを行方向配列入力に変換するため、この属性を設定しても効果はありません。

次の場合には、変換は実行されません。

- パラメーターを SQL_CLOB、SQL_BLOB、SQL_LONGVARBINARY、SQL_LONGVARGRAPHIC、SQL_DBCLOB、SQL_XML などの LOB データ・タイプにバインドする場合。
- SQLPutData() 関数および SQLParamData() 関数を呼び出して、値を SQL_DATA_AT_EXEC に設定してデータを INSERT 操作に渡すことによって、入力される実行時データ・パラメーターをバインドする場合。
- 内部バッファー内ですべてのアプリケーション・データを格納するためのスペースを利用できない場合。

SQL_ATTR_CONCURRENCY

カーソルの並行性を指定する 32 ビット整数値。

- SQL_CONCUR_READ_ONLY = カーソルは読み取り専用です。更新はできません。前方スクロールの静的なキー・セット・カーソルでサポートされます。
- SQL_CONCUR_LOCK = カーソルは、行を確実に更新できるロックのレベルのうち最低レベルのものを使用します。前方スクロールのキー・セット・カーソルでサポートされます。
- SQL_CONCUR_VALUES = カーソルは、値を比較し、オプティミスティック並行性制御を使用します。

静的な前方スクロール・カーソルについては、SQL_ATTR_CONCURRENCY のデフォルト値は、SQL_CONCUR_READ_ONLY です。キー・セット・カーソルのデフォルト値は、SQL_CONCUR_VALUES です。

この属性は、オープン・カーソルには指定できません。

SQL_ATTR_CURSOR_TYPE *属性* が SQL_ATTR_CONCURRENCY の現在の値をサポートしないタイプに変更されている場合、SQL_ATTR_CONCURRENCY の値は実行時に変更されることになり、SQLExecDirect() や SQLPrepare() を呼び出すと、警告が出されます。

SQL_ATTR_CONCURRENCY が SQL_CONCUR_READ_ONLY の値に設定されているときに、 SELECT FOR UPDATE ステートメントが実行されるとエラーが戻されます。SQL_ATTR_CONCURRENCY が、SQL_ATTR_CURSOR_TYPE の現在の値ではなく、SQL_ATTR_CURSOR_TYPE の何らかの値としてサポートされている値に変更される場合は、 SQL_ATTR_CURSOR_TYPE の値は実行時に変更され、SQLExecDirect() または SQLPrepare() を呼び出すと、 SQLSTATE 01S02 (オプション値が変更された) が出されます。

SQL_ATTR_CONCURRENCY の値が SQL_CONCUR_LOCK に設定されている場合、この値は以下の条件がすべて満たされたときに SQL_CONCUR_VALUES にプロモートされます。

- SQL_ROWSET_SIZE または SQL_ATTR_ROW_ARRAY_SIZE が 1 より大きい。
- データ・ソースが、DB2 Database for Linux, UNIX, and Windows サーバー上のデータベースである。
- **PATCH2** 構成キーワードが 73 に設定されている。

指定した並行性がデータ・ソースにサポートされていないと、CLI は別の並行性を代用し、 SQLSTATE 01S02 (オプション値が変更された) を戻します。代用の順序は、以下のようにカーソルのタイプによって異なります。

- 前方スクロール: SQL_CONCUR_LOCK が、 SQL_CONCUR_ROWVER および SQL_CONCUR_VALUES の代わりに使用される
- 静的: SQL_CONCUR_READ_ONLY だけが有効
- キー・セット: SQL_CONCUR_VALUES が、SQL_CONCUR_ROWVER の代わりに使用される

注: 以下の値も ODBC で定義されていますが、CLI ではサポートされません。

- SQL_CONCUR_ROWVER = カーソルは、オプティミスティック並行性制御を使用します。

SQL_ATTR_CURSOR_HOLD

この *StatementHandle* に関連したカーソルを COMMIT 操作前と同じ位置に保存するかどうか、また、アプリケーションがステートメントを再実行しなくてもフェッチできるようにするかどうかを指定する 32 ビット整数値。

- **SQL_CURSOR_HOLD_ON** (これはデフォルト値です)
- SQL_CURSOR_HOLD_OFF

StatementHandle が最初に割り当てられるときのデフォルト値は、SQL_CURSOR_HOLD_ON です。

このオプションは、この *StatementHandle* に関連したオープン・カーソルがある場合には設定できません。

CURSORHOLD CLI/ODBC 構成キーワードを使用して、デフォルト・カーソル保留モードを設定することもできます。

注: このオプションは、IBM 拡張機能です。

SQL_ATTR_CURSOR_SCROLLABLE

アプリケーションで必要とされるサポートのレベルを指定する 32 ビット整数。この属性を設定すると、後続の `SQLExecute()` および `SQLExecDirect()` の呼び出しが影響を受けます。以下の値がサポートされています。

- **SQL_NONSCROLLABLE** = 両方向スクロール・カーソルは、ステートメント・ハンドルで必要ではありません。アプリケーションがこのハンドルで `SQLFetchScroll()` を呼び出すと、 *FetchOrientation()* の有効値は SQL_FETCH_NEXT だけになります。この値はデフォルトです。
- SQL_SCROLLABLE = 両方向スクロール・カーソルが、ステートメント・ハンドルに必要です。 `SQLFetchScroll()` を呼び出すときに、アプリケーションは、順次モード以外のモードでカーソル位置を指定し、*FetchOrientation* の任意の有効な値を指定することができます。

SQL_ATTR_CURSOR_SENSITIVITY

ステートメント・ハンドル上のカーソルが、別のカーソルによる結果セットへの変更に影響を受けるかどうかを指定する 32 ビット整数。この属性を設定すると、後続の `SQLExecute()` および `SQLExecDirect()` の呼び出しが影響を受けます。以下の値がサポートされています。

- **SQL_UNSPECIFIED** = カーソル・タイプが何であるか、およびステートメント・ハンドル上のカーソルが、別のカーソルによる結果セットへの変更に影響を受けるかどうかは指定されません。ステートメント・ハンドル上のカーソルは、変更のいずれからも影響を受けないか、その一部、あるいは全部から影響を受けるようにすることができます。この値はデフォルトです。
- SQL_INSENSITIVE = ステートメント・ハンドル上のすべてのカーソルが示す結果セットには、別のカーソルがその結果セットに対して行った変更が反映されません。インセンシティブ・カーソルは、読み取り専用です。これは、読み取り専用である並行性のある静的カーソルと同じです。
- SQL_SENSITIVE = ステートメント・ハンドル上のすべてのカーソルは、別のカーソルによる結果セットへのすべての変更を可視にします。

SQL_ATTR_CURSOR_TYPE
カーソル・タイプを指定する 32 ビット整数値。以下の値がサポートされています。
- **SQL_CURSOR_FORWARD_ONLY** = カーソルは前方スクロールのみ可能です。これはデフォルトです。
- SQL_CURSOR_STATIC = 結果セット内のデータは、静的です。
- SQL_CURSOR_KEYSET_DRIVEN = CLI は純キー・セット・カーソルをサポートします。 SQL_KEYSET_SIZE ステートメント属性は無視されます。キー・セットのサイズを制限するには、アプリケーションは、SQL_ATTR_MAX_ROWS 属性を 0 以外の値に設定することによって、結果セットのサイズを制限する必要があります。
- SQL_CURSOR_DYNAMIC = 動的両方向スクロール・カーソルは、結果セットに対するすべての変更 (挿入、削除、および更新) を検出し、結果セットに対して挿入、削除、および更新を実行します。動的カーソルは、DB2 for z/OS バージョン 8.1 以降のサーバーにアクセスする場合のみサポートされます。

このオプションは、オープン・カーソルには指定できません。

指定したカーソル・タイプが、データ・ソースでサポートされていないと、CLI は別のカーソル・タイプを代用し、SQLSTATE 01S02 (オプション値が変更された) を戻します。混合または動的カーソルについては、CLI は、キー・セットによって操作されるカーソルまたは静的カーソルをこの順で代用します。

SQL_ATTR_DB2_NOBINDOUT
フェッチ操作中に、クライアントがデータ変換およびそれに関連する作業をいつ、どこで実行するかを指定する Boolean 値属性。この属性のデフォルト値は 0 (偽) です。 1 (真) に設定するのはフェデレーテッド・データベースに接続した場合だけにしてください。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_DEFERRED_PREPARE
対応する実行要求が発行されるまで、PREPARE 要求を据え置きにするかどうかを指定します。

- SQL_DEFERRED_PREPARE_OFF = 据え置き準備を無効にする。PREPARE 要求は、発行された時点で実行されます。
- **SQL_DEFERRED_PREPARE_ON** (デフォルト値) = 据え置き準備を有効にする。対応する実行要求が発行されるまで、PREPARE 要求の実行は据え置かれます。その後、ネットワーク・フローを最小化しパフォーマンスを改善するため、 2 つの要求が 2 つではなく 1 つのコマンド/応答のフローに結合されます。

  ターゲットの DB2 データベースまたは DDCS ゲートウェイが据え置き準備をサポートしていない場合、クライアントは、その接続の据え置き準備を無効にします。

注: 据え置き準備を有効にすると、通常は SQLCA の PREPARE ステートメントの SQLERRD(3) と SQLERRD(4) に戻される行およびコスト見積もりが、ゼロになる可能性があります。これは、これらの値を使用して SQL ステートメントを続行するかどうかを決定するユーザーにとっては重要です。

DEFERREDPREPARE CLI/ODBC 構成キーワードを使用して、デフォルトの据え置き準備モードを設定することもできます。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_EARLYCLOSE

最後のレコードがクライアントに送られた際に、クライアントのカーソルをクローズせずにサーバーの一時カーソルを自動的にクローズできるようにするかどうかを指定します。

- SQL_EARLYCLOSE_OFF = サーバーの一時カーソルを先にクローズしない。
- **SQL_EARLYCLOSE_ON** = サーバーの一時カーソルを先にクローズする (デフォルト値)。

  これによって、カーソルが既にクローズされたことを認識できるため、明示的にクローズするためのステートメントを発行しなくても済み、ネットワーク要求が節減されます。

  このオプションをオンにすると、小さい結果セットを多数使用するアプリケーションの処理速度が向上します。

  EARLYCLOSE フィーチャーは、カーソル・タイプが SQL_CURSOR_FORWARD_ONLY 以外である場合は使用されません。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_ENABLE_AUTO_IPD

IPD の自動移植を実行するかどうかを指定する 32 ビット整数値。

- SQL_TRUE = `SQLPrepare()` の呼び出し後、IPD の自動移植を行う。
- SQL_FALSE = `SQLPrepare()` の呼び出し後、IPD の自動移植を行わない。

ステートメント属性 SQL_ATTR_ENABLE_AUTO_IPD のデフォルト値は、SQL_ATTR_AUTO_IPD 接続属性値と同じです。

接続属性 SQL_ATTR_ AUTO_IPD が SQL_FALSE の場合、ステートメント属性 SQL_ATTR_ENABLE_AUTO_IPD は SQL_TRUE に設定できません。

SQL_ATTR_EXTENDED_INDICATORS
32 ビットの整数。この値により、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 ステートメント内での位置を示す必要がなくなります。この属性の値は次のとおりです。

- SQL_EXTENDED_INDICATOR_ENABLE: ユーザーは SQLBindParameter / SQLExtendedBind メソッドに SQL_UNASSIGNED と SQL_DEFAULT_PARAM を示す値を指定できます。
- SQL_EXTENDED_INDICATOR_NOT_SET (デフォルト): アプリケーションが SQL_UNASSIGNED と SQL_DEFAULT_PARAM を使用しようとする前にこれらの値が使用可能になっていないと、ユーザーは InvalidArgument 値エラーを受け取ります。
- DB2 for Linux, UNIX, and Windows および DB2 10 for z/OS のデータ・サーバーの拡張標識は、DB2 バージョン 9.7 フィックスパック 2 以降でサポートされます。DB2 for IBM i 7.1 のデータ・サーバーの拡張標識は、DB2 バージョン 9.7 フィックスパック 5 以降でサポートされます。

SQL_ATTR_FETCH_BOOKMARK_PTR
バイナリー数ブックマーク値を指すポインター。 SQL_FETCH_BOOKMARK に等しい *FetchOrientation* で SQLFetchScroll() を呼び出すと、 CLI はこのフィールドからブックマークをピックアップします。このフィールドは、デフォルトで NULL ポインターになります。

SQL_ATTR_IMP_PARAM_DESC
IPD へのハンドル。この属性の値は、ステートメントの初期割り当て時に割り当てられる記述子です。アプリケーションではこの属性を設定できません。

この属性の検索は、SQLGetStmtAttr() の呼び出しで行えますが、SQLSetStmtAttr() を呼び出して設定することはできません。

SQL_ATTR_IMP_ROW_DESC
IRD へのハンドル。この属性の値は、ステートメントの初期割り当て時に割り当てられる記述子です。アプリケーションではこの属性を設定できません。

この属性の検索は、SQLGetStmtAttr() の呼び出しで行えますが、SQLSetStmtAttr() を呼び出して設定することはできません。

SQL_ATTR_INFO_PROGRAMID
アプリケーションとステートメントを関連付ける最大長 80 バイトのユーザー定義文字ストリング。この属性を設定すると、DB2 UDB for z/OS バージョン 8 以降では、動的 SQL ステートメント・キャッシュに挿入されたステートメントにこの ID が関連付けられます。

この属性は、DB2 UDB for z/OS バージョン 8 以降または IBM Informix Dynamic Servers (IDS) にアクセスする CLI アプリケーションでのみサポートされます。

SQL_ATTR_INSERT_BUFFERING

この属性は、パーティション・データベース環境においてバッファリング挿入の最適化を有効にします。可能な値は、
SQL_ATTR_INSERT_BUFFERING_OFF (デフォルト)、
SQL_ATTR_INSERT_BUFFERING_ON、および
SQL_ATTR_INSERT_BUFFERING_IGD (重複は無視) です。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_INTERLEAVED_PUTDATA

この属性は、SQLParamData および SQLPutData を使用したインターリービング方式での LOB データの挿入を可能にします。例えば、以下のようにします。

```
// Set the attribute
SQLSetStmtAttr(hstmt,
               SQL_ATTR_INTERLEAVED_PUTDATA,
               TRUE,
               0);

//Bind the parameters with DATA_AT_EXEC indicator
blobInd = SQL_DATA_AT_EXEC;

cliRC = SQLBindParameter (hstmt,          /* statement handle      */
                          1,              /* parameter marker index */
                          SQL_PARAM_INPUT, /* it's input parameter  */
                          SQL_C_CHAR,      /* CLI variable is CHARACTER*/
                          SQL_CLOB,        /* table column is CLOB*/
                          10,              /* length of CLI variable */
                          0,              /* scale of decimal digits*/
                          &data1,          /* pointer to CLI variable*/
                          10,              /* buffer length          */
                          &blobInd);


cliRC = SQLBindParameter (hstmt,          /* statement handle      */
                          2,              /* parameter marker index */
                          SQL_PARAM_INPUT, /* it's input parameter  */
                          SQL_C_CHAR,      /* CLI variable is CHARACTER*/
                          SQL_CLOB,        /* table column is CLOB*/
                          10,              /* length of CLI variable */
                          0,              /* scale of decimal digits*/
                          &data2,          /* pointer to CLI variable*/
                          10,              /* buffer length          */
                          &blobInd);


SQLExecute (hstmt);
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data2
SQLPutData (hstmt, data2, strlen(data2));
valuePtr = (SQLPOINTER) 1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data1
SQLPutData (hstmt, data1, strlen(data2));
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data2
SQLPutData (hstmt, data2, strlen(data2));
valuePtr = (SQLPOINTER) 1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
//update buffer data1
```

```
SQLPutData (hstmt, data1, strlen(data2));

valuePtr = (SQLPOINTER) 0;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
```

この属性は、有効になっている SQLPutData 関数ストリーミングを無効にし、SQL_DATA_AT_EXEC でのデータが SQLParamData(0) でクローズされるまで各パラメーター値がクライアントでバッファーに入れられるようにします。

SQL_ATTR_INTERLEAVED_STREAM_PUTDATA

この属性は、関数ストリーミングを使用したインターリービング方式で、SQLParamData および SQLPutData を使用して LOB データを挿入できるようにします。ストリーミングは、パフォーマンスを高めるため、内部のステートメント・レベル・バッファーを介さずに、接続レベルのバッファーに LOB データを直接書き込みます。

SQLExecute、SQLParamData、または SQLPutData の間に新しい属性が設定されると、アプリケーションは、「関数のシーケンス・エラーです (CLI0125E)」のエラーを受け取ります。また、このエラーは、SQLParamData および SQLPutData の順序が正しくない場合にも返されます。

この属性が設定されたステートメント・ハンドルが接続内にある場合、SQL_ATTR_INTERLEAVED_STREAM_PUTDATA 属性が有効になっているステートメント・ハンドル用のサーバーへのデータ送信が完了しないうちに、接続バッファーを使用する操作が実行されると、同じ接続内の全ステートメント・ハンドルが、「関数のシーケンス・エラーです (CLI0125E)」のエラーを受け取ります。 SQL_ATTR_INTERLEAVED_STREAM_PUTDATA 属性が有効になっているステートメント・ハンドル用の SQLParamData および SQLPutData 呼び出しは、他のステートメント・ハンドルによって接続バッファーを使用する操作が実行される前に、すべて完了していなければなりません。ストリーミングを使用する場合の制限について詳しくは、453 ページの『StreamPutData CLI/ODBC 構成キーワード』を参照してください。

インターリーブド・パラメーター・セットの全パラメーターのデータの終わりを示すには、パラメーター番号 0 を指定して SQLParamData を呼び出します。アプリケーションは、パラメーター番号 0 を指定して SQLParamData を呼び出すことによって、全パラメーターのデータの終わりを明示的に示さなければなりません。

単一のパラメーターのデータの終わりを示すには、そのパラメーター番号の負の値を指定して SQLParamData を呼び出します。例えば、パラメーター番号 4 のデータ・ストリームの終わりを示す場合、アプリケーションは SQLParamData(-4) と指定する必要があります。アプリケーションは、負の値のパラメーター番号を使用して、パラメーターのデータの終わりを必ず示さなければなりません。アプリケーションが、ストリーム中のパラメーターのデータの終わりを示すと、CLIは、次のパラメーターのデータをストリームできるようになります。これによってパフォーマンスが向上します。

以下は、 LOB データのインターリービングについて、負の値のパラメーター番号を使用して単一パラメーターのデータの終わりを示す方法、および、パラメーター番号 0 を使用して全パラメーターのデータの終わりを示す方法の例です。

```
// Set the SQL_ATTR_INTERLEAVED_STREAM_PUTDATA attribute
SQLSetStmtAttr(hstmt, SQL_ATTR_INTERLEAVED_STREAM_PUTDATA, TRUE, 0);

//Bind the parameters with DATA_AT_EXEC indicator
blobInd = SQL_DATA_AT_EXEC;

//declare the statement handle with parameter marker
index value of 1,
//input parameter SQL_PARAM_INPUT, CLI variable type SQL_C_CHAR,
table column type CLOB,
//length of CLI variable 10, scale of decimal digits 10,
and DATA_AT_EXEC indicator
cliRC = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CLOB,
10, 0, &data1, 10, &blobInd);

//declare the next statement handle with
parameter marker index value of 2
cliRC = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CLOB,
10, 0, &data2, 10, &blobInd);

//declare the next statement handle with
parameter marker index value of 3
cliRC = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CLOB,
10, 0, &data3, 10, &blobInd);

SQLExecute (hstmt);

//make parameter 2 active
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 2
SQLPutData (hstmt, data2, strlen(data2));

//make parameter 1 active
valuePtr = (SQLPOINTER) 1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

// stream data for parameter 1
SQLPutData (hstmt, data1, strlen(data2));

//make parameter 2 active
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 2
SQLPutData (hstmt, data2, strlen(data2));

//make parameter 3 active
valuePtr = (SQLPOINTER) 3;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 3
SQLPutData (hstmt, data1, strlen(data2));

//end of data for parameter 1
valuePtr = (SQLPOINTER) -1;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
```

```
//make parameter 2 active
valuePtr = (SQLPOINTER) 2;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//stream the buffered data for parameter 2
SQLPutData (hstmt, data2, strlen(data2));

//make parameter 3 active valuePtr = (SQLPOINTER) 3;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

//buffer data for parameter 3
SQLPutData (hstmt, data1, strlen(data2));

//end of data for parameter 3
valuePtr = (SQLPOINTER) -3;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);

// indicate end of data for all parameters.
// CLI streams the buffered data for all parameters
valuePtr = (SQLPOINTER) 0;
SQLParamData (hstmt, (SQLPOINTER *)&valuePtr);
```

SQL_ATTR_KEYSET_SIZE

CLI は純キー・セット・カーソルをサポートしているので、 SQL_KEYSET_SIZE ステートメント属性は無視されます。キー・セットのサイズを制限するには、アプリケーションは、 SQL_ATTR_MAX_ROWS 属性を 0 以外の値に設定することによって、結果セットのサイズを制限する必要があります。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_INFO

db2LoadStruct 型の構造体へのポインター。 db2LoadStruct 構造体は、 CLI LOAD 中に使用すべき適用可能なすべての LOAD オプションを指定するのに使用します。注意が必要な点として、このポインターおよび組み込まれたポインターすべては、 SQL_ATTR_USE_LOAD_API ステートメント属性が設定されたときから、それがオフになるまでのすべての CLI 関数呼び出し中、有効でなければなりません。このため、このポインターと組み込まれたポインターすべては、ローカルに宣言された構造体ではなく、動的に割り振られたメモリーを指すようにすることをお勧めします。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_MODIFIED_BY

CLI LOAD 時に使用されるファイル・タイプ修飾子オプションを指定する char ストリングへのポインター。

SQL_ATTR_LOAD_ROWS_COMMITTED_PTR

処理された合計行数を表す整数へのポインター。この値は、正常にロードされ、データベースにコミットされた行数と、スキップおよび拒否された行数の合計と等しくなります。この整数は、32 ビット・プラットフォームでは 32 ビット、 64 ビット・プラットフォームでは 64 ビットです。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_ROWS_DELETED_PTR
削除された重複行数を表す整数へのポインター。この整数は、32 ビット・プラットフォームでは 32 ビット、 64 ビット・プラットフォームでは 64 ビットです。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_ROWS_LOADED_PTR
ターゲット表にロードされた行数を表す整数へのポインター。この整数は、32 ビット・プラットフォームでは 32 ビット、 64 ビット・プラットフォームでは 64 ビットです。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_ROWS_READ_PTR
読み取られた行数を表す整数へのポインター。この整数は、32 ビット・プラットフォームでは 32 ビット、 64 ビット・プラットフォームでは 64 ビットです。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_ROWS_REJECTED_PTR
ロードできなかった行数を表す整数へのポインター。この整数は、32 ビット・プラットフォームでは 32 ビット、 64 ビット・プラットフォームでは 64 ビットです。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOAD_ROWS_SKIPPED_PTR
CLI LOAD 操作開始の前にスキップされた行数を表す整数へのポインター。この整数は、32 ビット・プラットフォームでは 32 ビット、 64 ビット・プラットフォームでは 64 ビットです。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_LOB_CACHE_SIZE
LOB の最大キャッシュ・サイズ (バイト単位) を指定する、32 ビットの符号なし整数。デフォルトでは、LOB はキャッシュに入れられません。

詳しい使用情報については、 411 ページの『LOBCacheSize CLI/ODBC 構成キーワード』を参照してください。

SQL_ATTR_MAX_LENGTH
単一文字またはバイナリー列から取り出せるデータの最大量に対応する 32 ビット整数値。

注: SQL_ATTR_MAX_LENGTH を使ってデータを切り捨ててはなりません。データを切り捨てるには、 `SQLBindCol()` または `SQLGetData()` の *BufferLength* 引数を使用しなければなりません。
SQL_ATTR_MAX_LENGTH に指定した値が使用できるデータ量よりも小さいためにデータが切り捨てられる場合、`SQLGetData()` の呼び出しまたはフ

ェッチは SQL_SUCCESS_WITH_INFO および SQLSTATE **01**004 (データ切り捨て) ではなく、SQL_SUCCESS を戻します。
SQL_ATTR_MAX_LENGTH のデフォルト値は **0** です。 0 の場合は、CLI は文字またはバイナリー形式の使用できるすべてのデータを戻そうとします。

SQL_ATTR_MAX_LOB_BLOCK_SIZE
LOB または XML データ・ブロックの最大サイズを示す 32 ビット符号なし整数。正の整数を、最高 2 147 483 647 まで指定してください。デフォルト設定の 0 は、LOB または XML データ・ブロックのデータ・ブロック・サイズに制限がないことを示します。

データ検索時にサーバーは、最大ブロック・サイズに到達している場合でも、現在行に関するすべての情報をクライアントへの応答に含めます。

MaxLOBBlockSize と **db2set** レジストリー変数 **DB2_MAX_LOB_BLOCK_SIZE** の両方が指定されている場合、**MaxLOBBlockSize** の値が使用されます。

422 ページの『MaxLOBBlockSize CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_MAX_ROWS
照会からアプリケーションに戻す最大行数に対応する 32 ビット整数値。
SQL_ATTR_MAX_ROWS のデフォルト値は **0** です。 0 の場合は、すべてのデータが戻されます。

SQL_ATTR_METADATA_ID
このステートメント属性は ODBC で定義されていますが、CLI ではサポートされません。この属性を設定または取得しようとすると、 SQLSTATE は HYC00 (ドライバーが機能しない) になります。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_NOSCAN
CLI が SQL をスキャンしてエスケープ節のストリングを探すかどうかを指定する 32 ビット整数値。次の 2 つの有効値があります。
- **SQL_NOSCAN_OFF** - SQL ストリングをスキャンして、エスケープ節シーケンスを探します。これはデフォルトです。
- SQL_NOSCAN_ON - SQL ストリングをスキャンしてエスケープ節を探しません。すべてサーバーに直接送られ処理されます。

このアプリケーションは、送信する SQL ストリング内にベンダー・エスケープ・シーケンスを使用することがない場合にスキャンをオフにすることを選択できます。スキャンをオフにすると、スキャンに関連する処理使用量が一部抑えられます。

SQL_ATTR_OPTIMIZE_FOR_NROWS
32 ビット整数値。 n を 0 より大きい整数に設定すると、 "OPTIMIZE FOR n ROWS" 節がすべての選択ステートメントに付加されます。 0 (デフォルト値) に設定すると、この節は追加されません。

OPTIMIZEFORNROWS CLI/ODBC 構成キーワードを使用してデフォルト値を設定することもできます。

SQL_ATTR_OPTIMIZE_SQLCOLUMNS
この属性は、使用すべきでない属性となりました。

SQL_ATTR_PARAM_BIND_OFFSET_PTR
動的パラメーターのバインドを変更するためにポインターに追加されるオフセットを示す 32 ビット整数 * 値。このフィールドがヌル以外の場合、CLI はポインターの参照を解除し、参照解除された値を記述子レコード (SQL_DESC_DATA_PTR、SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR) の各据え置きフィールドに追加し、その結果のポインター値を実行時に使用します。これはデフォルトで NULL に設定されます。

バインド・オフセットは、常に SQL_DESC_DATA_PTR、SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR フィールドに直接追加されます。オフセットを別の値に変更した場合、新しい値が記述子フィールドの値に直接追加されます。新しいオフセットは、以前のオフセットを加えたフィールド値に追加されることはありません。

このステートメント属性を設定すると、 APD ヘッダーに SQL_DESC_BIND_OFFSET_PTR フィールドが設定されます。

SQL_ATTR_PARAM_BIND_TYPE
バインド方向を動的パラメーターに使用することを示す 32 ビット整数値。

このフィールドを **SQL_PARAM_BIND_BY_COLUMN** (デフォルト) に設定して、列方向のバインドを選択します。

行方向のバインドを選択するには、このフィールドを構造体の長さか、または、動的パラメーターのセットにバインドされるバッファーのインスタンス長に設定します。この長さには、バインドされるパラメーターのすべてに対するスペースと、構造体やバッファーの埋め込みが入っていなければなりません。これは、バインドされるパラメーターのアドレスを指定の長さで増分した際に、必ず結果が次のパラメーター・セットの同じパラメーターの先頭を指し示すようにするためです。 ANSI C の sizeof 演算子を使用する場合、この動作は保証されます。

このステートメント属性を設定すると、 APD ヘッダーに SQL_DESC_BIND_TYPE フィールドが設定されます。

SQL_ATTR_PARAM_OPERATION_PTR
SQL ステートメントの実行中にパラメーターを無視するかどうかを指定するのに使用される 16 ビット無符号整数値の配列を指す 16 ビット無符号整数 * 値。それぞれの値は、SQL_PARAM_PROCEED (パラメーターを実行する) か、または SQL_PARAM_IGNORE (パラメーターを無視する) に設定します。

APD の SQL_DESC_ARRAY_STATUS_PTR が指し示す配列の状況値を SQL_PARAM_IGNORE に設定することによって、処理中にパラメーターのセットを無視することができます。状況値が SQL_PARAM_PROCEED に設定されているか、配列のどのエレメントも設定されていなければ、パラメーターのセットが処理されます。

このステートメント属性は NULL ポインターに設定可能です。その場合、CLI はパラメーター状況値を戻しません。この属性はいつでも設定可能ですが、設定した値は、次に SQLExecDirect() または SQLExecute() を呼び出すまで使用されません。

このステートメント属性を設定すると、 APD に SQL_DESC_ARRAY_STATUS_PTR フィールドが設定されます。

SQL_ATTR_PARAM_STATUS_PTR

SQLExecDirect() または SQLExecute() の呼び出し後、パラメーター値の各行ごとの状況に関する情報の入った UWORD 値の配列を示す 16 ビット無符号整数 * 値。このフィールドは、SQL_ATTR_PARAMSET_SIZE が 1 より大きい場合にのみ使います。

状況値には以下の値が入ります。

- SQL_PARAM_SUCCESS: SQL ステートメントは、このパラメーターのセットに対して正常に実行されました。
- SQL_PARAM_SUCCESS_WITH_INFO: SQL ステートメントは、このパラメーターのセットに対して正常に実行されました。ただし、診断データ構造体の中に警告情報があります。
- SQL_PARAM_ERROR: このパラメーターのセットを処理中にエラーがありました。診断データ構造体の中に追加のエラー情報があります。
- SQL_PARAM_UNUSED: このパラメーター・セットは使用できませんでした。前のパラメーター・セットのいずれかでエラーが発生し、処理が打ち切られたことが原因とみられます。
- SQL_PARAM_DIAG_UNAVAILABLE: CLI は、パラメーターの配列を一体構造の単位として扱うため、このレベルのエラー情報を生成しません。

このステートメント属性は NULL ポインターに設定可能です。その場合、CLI はパラメーター状況値を戻しません。この属性はいつでも設定可能ですが、設定した値は、次に SQLFetch()、SQLFetchScroll()、または SQLSetPos() を呼び出すまで使用されません。

このステートメント属性を設定すると、 IPD ヘッダーに SQL_DESC_ARRAY_STATUS_PTR フィールドが設定されます。

SQL_ATTR_PARAMOPT_ATOMIC

SQLParamOptions() を使用してパラメーター・マーカーの複数値を指定した場合に、基礎処理の実行を ATOMIC と NOT-ATOMIC コンパウンド SQL のどちらを介して行うかを判別する 32 ビット整数値。以下の値を指定することができます。

- **SQL_ATOMIC_YES** - 基礎処理で ATOMIC コンパウンド SQL を使用します。ターゲット・データベースが ATOMIC コンパウンド SQL をサポートする場合は、これがデフォルト値です。
- SQL_ATOMIC_NO - 基礎処理で NON-ATOMIC コンパウンド SQL を使用します。

ATOMIC コンパウンド SQL をサポートしないサーバーへの接続時に SQL_ATOMIC_YES を指定するとエラーになります (SQLSTATE は **S1**C00 です)。

SQL_PARC_BATCH が SQL_PARC_BATCH_ENABLE に設定されているときに SQL_ATOMIC_YES を指定すると、CLI0150E エラー・メッセージが

返されます。SQL_PARC_BATCH を SQL_PARC_BATCH_ENABLE に設定する場合は、SQL_ATOMIC_NO を指定する必要があります。

SQL_ATTR_PARAMS_PROCESSED_PTR
現在行の番号を戻すべきバッファーを示す 32 ビット無符号整数 * レコード・フィールド。パラメーターの各行が処理されていく際に、これはその行番号に設定されます。これが NULL ポインターであれば、行番号は返されません。

このステートメント属性を設定すると、 IPD ヘッダーに SQL_DESC_ROWS_PROCESSED_PTR フィールドが設定されます。

この属性が指しているバッファーに入っている SQLExecDirect() または SQLExecute() が SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合、そのバッファーの内容は未定義になります。

SQL_ATTR_PARAMSET_SIZE
各パラメーターごとの値の数を指定する 32 ビット無符号整数値。SQL_ATTR_PARAMSET_SIZE が 1 より大きい場合、APD の SQL_DESC_DATA_PTR、 SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR は配列を示します。各配列のカーディナリティーは、このフィールドの値と同等です。

このステートメント属性を設定すると、APD ヘッダーに SQL_DESC_ARRAY_SIZE フィールドが設定されます。

DB2 バージョン 9.7 フィックスパック 6 以降、信頼できるプロシージャー本体に含まれる、SQL_ATTR_PARAMSET_SIZE を使用した配列入力は、サポートされるようになっています。

SQL_ATTR_PREFETCH
この属性は、使用すべきでない属性となりました。

SQL_ATTR_QUERY_OPTIMIZATION_LEVEL
SQLPrepare()、SQLExtendedPrepare()、または SQLExecDirect() の次の呼び出しで照会最適化レベルが使用されるように設定する 32 ビット整数値。

使用がサポートされている値は -1 (デフォルト)、0、1、2、3、5、7、および 9 です。

IDS データ・サーバーの場合、SQL_ATTR_QUERY_OPTIMIZATION_LEVEL ステートメント属性を使用して最適化レベルを設定することはできません。代わりに Informix オプティマイザー・ディレクティブを使用する必要があります。詳しくは、 オプティマイザ ディレクティブを参照してください。

SQL_ATTR_QUERY_TIMEOUT
SQL ステートメントまたは XQuery 式の実行を待機し始めてから、その実行を打ち切ってアプリケーションに戻るまでの秒数を示す 32 ビット整数値。このオプションを設定しておけば、長期実行照会を終了するために使用することができます。 **0** のデフォルト値は、サーバーが SQL ステートメントの実行を完了するのを CLI は無期限に待機することを意味します。CLI は、マルチスレッド化をサポートするどのプラットフォームでも非ゼロ値をサポートします。

ネイティブ割り込みサポートのないサーバー (DB2 for z/OS および OS/390、バージョン 7 以前、および DB2 for i など) に対してこの属性を使用する場合、サーバーに対応する DCS データベース項目のカタログ時に INTERRUPT_ENABLED オプションが設定されていなければなりません。

INTERRUPT_ENABLED オプションが設定されており、この属性がゼロ以外の値に設定されると、DB2 for i サーバーは接続を除去し、作業単位をロールバックします。アプリケーションは、サーバーへの接続が終了したことを示す、SQL30081N エラーを受け取ります。アプリケーションが追加のデータベース要求を処理するには、アプリケーションがデータベース・サーバーとの新規接続を確立する必要があります。

SQL_ATTR_QUERY_TIMEOUT のために LOAD が中断されることもあります。この場合は、SQL0952N ではなく SQL3005N が返されます。

SQL_ATTR_REOPT

特殊レジスターまたはパラメーター・マーカーを含む SQL ステートメントに対して照会最適化を有効にする 32 ビット整数値。コンパイラーによって選択されるデフォルトの推定値の代わりに、特殊レジスターまたはパラメーター・マーカーに対して照会実行時に使用できる値を使用することによって、最適化が生じます。属性の有効値は、次のとおりです。

- 2 = SQL_REOPT_NONE。これはデフォルトです。照会の実行時に照会最適化は行われません。コンパイラーによって選択されるデフォルトの推定値が、特殊レジスターまたはパラメーター・マーカーに対して使用されます。デフォルトの NULLID パッケージ・セットは、動的 SQL ステートメントの実行に使用されます。
- 3 = SQL_REOPT_ONCE。照会最適化は、照会実行時に一度、初めて照会を実行するときに生じます。 NULLIDR1 パッケージ・セットが使用されますが、これは REOPT ONCE バインド・オプションとバインドされています。
- 4 = SQL_REOPT_ALWAYS。照会最適化または再最適化は、照会の実行時に、毎回必ず生じます。 NULLIDRA パッケージ・セットが使用されますが、これは REOPT ALWAYS バインド・オプションとバインドされています。

NULLIDR1 および NULLIDRA は予約済みパッケージ・セット名で、使用時には REOPT ONCE および REOPT ALWAYS が暗黙指定されます。これらのパッケージ・セットは、これらのコマンドで明示的に作成する必要があります。

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

SQL_ATTR_REOPT と SQL_ATTR_CURRENT_PACKAGE_SET を同時に指定することはできません。したがって、一方を設定すると、他方は許可されません。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_RETRIEVE_DATA

32 ビット整数値。

- **SQL_RD_ON** = SQLFetchScroll() および DB2 CLI v5 以降の SQLFetch() では、カーソルを指定のロケーションに置いた後でデータを検索します。これはデフォルトです。
- SQL_RD_OFF = SQLFetchScroll() および DB2 CLI v5 以降の SQLFetch() では、カーソルを指定の位置に置いた後でデータを検索しません。

アプリケーションは、SQL_RETRIEVE_DATA を SQL_RD_OFF に設定すれば、行検索に関連したリソースの使用量を増やさずに、行が存在するかどうかの検査や、行のブックマークの検索を行うことができます。

SQL_ATTR_RETURN_USER_DEFINED_TYPES
SQLDescribeCol() などの関数によってユーザー定義タイプの列について照会された場合に、そのユーザー定義タイプの列がユーザー定義タイプとして報告されるか、それともその基になる基本タイプとして報告されるかを指定する Boolean 値属性。デフォルト値は 0 (偽) であり、列は基になる基本タイプとして報告されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_ROW_ARRAY_SIZE
行セット内の行数を指定する 32 ビット整数値。これは、 SQLFetch() または SQLFetchScroll() への呼び出しのたびに戻される行数です。デフォルト値は 1 です。

指定した行セット・サイズが、データ・ソースでサポートされる最大行セット・サイズを超えると、 CLI はその値を代用し、SQLSTATE 01S02 (オプション値が変更された) を戻します。

このオプションは、オープン・カーソルにも指定できます。

このステートメント属性を設定すると、 ARD ヘッダーに SQL_DESC_ARRAY_SIZE フィールドが設定されます。

SQL_ATTR_ROW_BIND_OFFSET_PTR
列データのバインドを変更するためにポインターに追加されるオフセットを示す 32 ビット整数 * 値。このフィールドが非 NULL の場合、CLI はポインターを参照解除し、その参照解除された値を記述子レコード (SQL_DESC_DATA_PTR、 SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR) の各据え置きフィールドに追加し、バインド時に新しいポインター値を使用します。これはデフォルトで NULL に設定されます。

このステートメント属性を設定すると、 ARD ヘッダーに SQL_DESC_BIND_OFFSET_PTR フィールドが設定されます。

SQL_ATTR_ROW_BIND_TYPE
関連ステートメントで SQLFetch() または SQLFetchScroll() を呼び出すときに使用するバインド方向を設定するための 32 ビット整数値。引数 **ValuePtr* に定義済み定数 SQL_BIND_BY_COLUMN を指定すると、列方向のバインドが選択されます。構造体の長さまたは列のバインド先となるバッファーのインスタンスを指定する **ValuePtr* に値を指定すると、行方向のバインドが選択されます。

ValuePtr* に指定される長さには、バインドされる列のすべてに対するスペースと、構造体やバッファーの埋め込みが入っていなければなりません。これは、バインドされる列のアドレスを指定の長さで増分した際に、必ず結果が次の行の同じ列の先頭を示すようにするためです。ANSI C の構造体または共用体で **sizeof 演算子を使用する場合、この動作は保証されます。

列方向のバインドは、 SQLFetch() および SQLFetchScroll() のデフォルトのバインド方向です。

このステートメント属性を設定すると、 ARD ヘッダーに SQL_DESC_BIND_TYPE フィールドが設定されます。

SQL_ATTR_ROW_NUMBER
結果セット全体の現在行の番号を示す 32 ビット整数値。現在行の番号を判別できないか、現在行がない場合、CLI は 0 を戻します。

この属性の検索は、SQLGetStmtAttr() の呼び出しで行えますが、SQLSetStmtAttr() を呼び出して設定することはできません。

SQL_ATTR_ROW_OPERATION_PTR
SQLSetPos() を使用するバルク操作時に行を無視するための UDWORD 値の配列を示す 16 ビット無符号整数 * 値。それぞれの値は、SQL_ROW_PROCEED (バルク操作時に行を含める) または SQL_ROW_IGNORE (バルク操作時に行を除外する) に設定されます。

このステートメント属性は NULL ポインターに設定可能です。その場合、CLI は行状況値を戻しません。この属性はいつでも設定可能ですが、設定した値は、次に SQLFetch()、SQLFetchScroll()、または SQLSetPos() を呼び出すまで使用されません。

このステートメント属性を設定すると、 ARD に SQL_DESC_ARRAY_STATUS_PTR フィールドが設定されます。

SQL_ATTR_ROW_STATUS_PTR
SQLFetch() または SQLFetchScroll() の呼び出し後、行状況値の入った UWORD 値の配列を示す 16 ビット無符号整数 * 値。この配列のエレメントの数は、行セット内にある行の数と同じです。

このステートメント属性は NULL ポインターに設定可能です。その場合、CLI は行状況値を戻しません。この属性はいつでも設定可能ですが、設定した値は、次に SQLFetch()、SQLFetchScroll()、または SQLSetPos() を呼び出すまで使用されません。

このステートメント属性を設定すると、 IRD ヘッダーに SQL_DESC_ARRAY_STATUS_PTR フィールドが設定されます。

SQL_ATTR_ROWS_FETCHED_PTR
これは、 SQLFetch() または SQLFetchScroll() への呼び出し後、フェッチした行数を戻すべきバッファーを示す 32 ビット無符号整数 * 値。

このステートメント属性を設定すると、 IRD ヘッダーに SQL_DESC_ROWS_PROCESSED_PTR フィールドが設定されます。

SQLExtendedFetch() の呼び出し時、この属性は CLI により *RowCountPtr* 配列にマップされます。

SQL_ATTR_ROWCOUNT_PREFETCH

この属性は、CLI が行数を判別することを可能にし、結果セット全体をプリフェッチできるようにします。この属性の値は次のうちの 1 つとなります。

- **0 (デフォルト)**: オフ
- **1**: オン

SQL_ATTR_ROWCOUNT_PREFETCH を 0 に設定し、スクロール不能の SELECT 専用カーソルを使用して SQLRowCount() を呼び出した場合、すべてのデータがフェッチされるまで行数は使用不可であるため、この関数は *RowCountPtr* の内容を -1 に設定します。

SQL_ATTR_ROWCOUNT_PREFETCH を 1 に設定し、順方向の SELECT 専用カーソルを使用して SQLRowCount() を呼び出した場合、以下の動作が見られます。

- SELECT * FROM INSERT | UPDATE | DELETE ステートメントを前方スクロール・カーソルで使用する場合、行カウントは SELECT ステートメントに由来します。これは、SQL_ATTR_ROWCOUNT_PREFETCH 属性を設定していない場合にこれらのカーソルに設定される、影響を受けた行のカウントとは異なります。
- すべてのカーソル・データがプリフェッチされます。この場合、サーバーとの間で複数回の往復が必要になったり、クライアントに相当量のメモリーが必要になったりする可能性があります。
- プリフェッチされたデータは廃棄されず、アプリケーションのフェッチ要求に応じるために使用されます。

両方向スクロール・カーソルは行カウントを提供できるため、この属性は適用されません。

この属性は、ステートメントを準備する前に指定してください。

制約事項: SQL_ATTR_ROWCOUNT_PREFETCH は、カーソルに LOB または XML が含まれている場合はサポートされません。

SQL_ROWSET_SIZE

CLI アプリケーションは現在、SQLExtendedFetch() ではなく SQLFetchScroll() を使用する必要があります。また、アプリケーションでは行セットに行数を設定するために、ステートメント属性として SQL_ATTR_ROW_ARRAY_SIZE を使用してください。

行セット内の行数を指定する 32 ビット整数値。行セットは、SQLExtendedFetch() の呼び出しのたびに戻される行の配列です。デフォルト値は **1** ですが、これは単一の SQLFetch() 呼び出しを行うことと同等です。このオプションは、カーソルがオープンであるときでも指定することができ、次の SQLExtendedFetch() 呼び出しで有効になります。

SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)

このステートメント属性は、CLI ではサポートされませんが、 ODBC により定義されています。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_STMT_CONCENTRATOR
リテラル値を含む動的ステートメントがステートメント・キャッシュを使用するかどうかを指定します。

- SQL_STMT_CONCENTRATOR_OFF - ステートメント・コンセントレーターの動作は無効です。
- SQL_STMT_CONCENTRATOR_WITH_LITERALS - サーバーがサポートしている状況の場合、ステートメント・コンセントレーターが有効になり、リテラル値を含む動的ステートメントはステートメント・キャッシュを使用します。 例えば、ステートメントにパラメーター・マーカー、名前付きパラメーター・マーカー、またはリテラルとパラメーター・マーカーおよび名前付きパラメーター・マーカーの組み合わせが含まれている場合、ステートメント・コンセントレーターは有効になりません。

451 ページの『StmtConcentrator CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

注: DB2® for z/OS® サーバーのバージョン 10 より前のサーバーでこの属性を使用すると、要求は無視されます。

SQL_ATTR_STMTTXN_ISOLATION
SQL_ATTR_TXN_ISOLATION を参照してください。

SQL_ATTR_STREAM_GETDATA
`SQLGetData()` 関数のデータ出力ストリームが最適化されるかどうかを示す 32 ビット符号なし整数。値は以下のとおりです。

- **0 (デフォルト)**: CLI はクライアント上のすべてのデータをバッファーに入れます。
- 1: データをバッファーに入れる必要のないアプリケーションで Dynamic Data Format をサポートするサーバー上のデータを照会している場合は、1 を指定することによって、データをバッファーに入れる必要がないことを示してください。 CLI クライアントは、データ出力ストリームを最適化します。

Dynamic Data Format がサーバーによってサポートされない場合、このキーワードは無視されます。

`StreamGetData` が 1 に設定されている場合、出力バッファーに入れて戻すためにまだ使用できるバイト数を CLI が判別できない場合、切り捨て発生時に `SQLGetData()` は長さとして SQL_NO_TOTAL (-4) を戻します。それ以外の場合、`SQLGetData()` は、まだ使用できるバイト数を戻します。

452 ページの『StreamGetData CLI/ODBC 構成キーワード』の設定は、この動作を指定する別の方法です。

SQL_ATTR_TXN_ISOLATION
現行の *StatementHandle* にトランザクション分離レベルを設定する 32 ビット整数値。

このオプションは、このステートメント・ハンドルに関するオープン・カーソルがある場合には設定できません (SQLSTATE **24**000)。

値 SQL_ATTR_STMTTXN_ISOLATION は、SQL_ATTR_TXN_ISOLATION と同義です。ただし、ODBC Driver Manager は、ステートメント・オプシ

ョンとしての SQL_ATTR_TXN_ISOLATION の設定を拒否するので、各ステートメントごとにトランザクション分離レベルを設定しなければならない ODBC アプリケーションでは、 SQLSetStmtAttr() 呼び出しで、代わりに明示定数 SQL_ATTR_STMTTXN_ISOLATION を使用する必要があります。

TXNISOLATION CLI/ODBC 構成キーワードを使用してデフォルト・トランザクション分離レベルを設定することもできます。

この属性 (または対応するキーワード) を使用できるのは、ステートメント・ハンドル用のデフォルトの分離レベルが使用される場合だけです。一方、アプリケーションがステートメント・ハンドル用の分離レベルを設定していた場合、この属性には効力はありません。

注: これは、ステートメント・レベルでこのオプションを設定するための IBM 拡張機能です。

SQL_ATTR_USE_BOOKMARKS
アプリケーションがカーソルでブックマークを使用するかどうかを指定する 32 ビット整数値。

- **SQL_UB_OFF** = Off (デフォルト値)
- SQL_UB_VARIABLE = アプリケーションはカーソルでブックマークを使用し、 CLI は、サポートされていれば可変長のブックマークを提供します。

カーソルと一緒にブックマークを使用するには、アプリケーションは、カーソルのオープン前に SQL_UB_VARIABLE 値付きのオプションを指定する必要があります。

SQL_ATTR_USE_LOAD_API
データの挿入時に LOAD ユーティリティーが正規の CLI 配列挿入に取って代わるかどうかを示す 32 ビット整数。以下の値を指定することができます。

SQL_USE_LOAD_OFF
(デフォルト) 正規の CLI 配列挿入を使ってデータを挿入します。

SQL_USE_LOAD_INSERT
LOAD ユーティリティーを使用して、表中の既存のデータに追加します。

SQL_USE_LOAD_REPLACE
LOAD ユーティリティーを使用して、表中の既存のデータを置き換えます。

SQL_USE_LOAD_RESTART
以前に失敗した CLI LOAD 操作を再開します。以前の CLI LOAD 操作が失敗したのが行が挿入されている間である (つまり、SQL_ATTR_USE_LOAD_API ステートメント属性が SQL_USE_LOAD_OFF に設定される前) 場合、 CLI LOAD フィーチャーはアクティブのままで、後続の行は CLI LOAD ユーティリティーによって挿入されます。そうでなく、操作が失敗したのが CLI LOAD がオフにされている間である場合、再開されたロードの完了後、通常の CLI 配列の挿入が再開されます。

SQL_USE_LOAD_TERMINATE
以前に失敗した CLI LOAD 操作をクリーンアップし取り消します。ステートメント属性をこの値に設定した後、通常の CLI 配列挿入を再開します。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

注: バージョン 9.7 フィックスパック 4 以降、この属性は SQL_ATTR_ASYNC_ENABLE と一緒に使用することもできます。

SQL_ATTR_XML_DECLARATION
XML データが暗黙的にシリアライズされるときに XML 宣言のエレメントが XML データに追加されることを示す 32 ビット符号なし整数。この属性は XMLSERIALIZE 関数の結果に影響を与えません。

この属性は、オープン・カーソルが関連付けられていないステートメント・ハンドルでのみ指定できます。ステートメント・ハンドル上にオープン・カーソルがある間にこの属性の値を更新しようとすると、CLI0126E (SQLSTATE HY011) エラーが出され、値は未変更のままになります。

この属性を、必要な各コンポーネントの合計に設定します。

0 出力バッファーに追加される宣言またはバイト・オーダー・マーク (BOM) はありません。

1 ターゲット・エンコードが UTF-16 または UTF-32 の場合、該当するエンディアン (リトル・エンディアンまたはビッグ・エンディアン) のバイト・オーダー・マーク (BOM) が出力バッファーの前に付加されます。 (UTF-8 BOM がある場合でも、ターゲット・エンコードが UTF-8 であっても、DB2 は UTF-8 BOM を生成しません。)

2 XML バージョンだけを含む最小の XML 宣言が生成されます。

4 ターゲット・エンコードを識別するエンコード属性が、生成された XML 宣言に追加されます。そのため、この設定が有効になるのは、この属性の値を計算する際に設定 2 も含まれている場合に限ります。

SQLSetStmtAttr() または SQLSetStmtOption() を使用してその他の値を設定しようとすると、CLI0191E (SQLSTATE HY024) エラーが出され、値は未変更のままになります。

デフォルト設定は 7 です。これは、XML バージョンとエンコード属性を含む BOM および XML 宣言が暗黙的なシリアライズ中に生成されることを示します。

この属性は接続ハンドルで指定することもでき、値の変更後に割り振られるステートメント・ハンドルに影響を与えます。既存のステートメント・ハンドルは元の値を保持します。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

SQL_ATTR_XQUERY_STATEMENT
現行のステートメント・ハンドルに関連したステートメントが XQuery 式か、SQL ステートメントか、または照会であるかを指定する 32 ビット整

数値。これは、XQuery 式の前に "XQUERY" キーワードを付けない CLI アプリケーションで使用することができます。以下の値がサポートされています。

SQL_TRUE
現行のステートメント・ハンドルで実行される次のステートメントが XQuery 式として処理されます。サーバーが XQuery をサポートしない場合、この属性を SQL_TRUE に設定すると、警告 CLI0005W (SQLSTATE 01S02) が出され、属性の値は未変更になります。

SQL_FALSE (デフォルト)
現行のステートメント・ハンドルで実行される次のステートメントが SQL ステートメントとして処理されます。

この属性は、次の `SQLPrepare()` または `SQLExecDirect()` 関数呼び出しで実施されます。

IDS データ・サーバーにアクセスする場合、この属性はサポートされません。

第 5 章 記述子の値

記述子 FieldIdentifier 引数の値 (CLI)

FieldIdentifier 引数は、設定する記述子フィールドを示します。記述子には、ヘッダー・フィールド (次の項で説明) からなる記述子ヘッダーと、レコード・フィールド (この後の項で説明) からなる 0 個以上の記述子レコードとが含まれています。

ヘッダー・フィールド

各記述子には、以下のフィールドからなるヘッダーがあります。

SQL_DESC_ALLOC_TYPE [すべて] この読み取り専用の SQLSMALLINT ヘッダー・フィールドには、記述子が CLI によって自動的に割り当てられたか、あるいはアプリケーションにより明示的に割り当てられたかが指定されます。アプリケーションはこのフィールドを取得することはできますが、変更はできません。記述子が自動的に割り当てられた場合には、このフィールドは SQL_DESC_ALLOC_AUTO に設定されます。また、アプリケーションにより明示的に割り当てられた場合には SQL_DESC_ALLOC_USER に設定されます。

SQL_DESC_ARRAY_SIZE [アプリケーション記述子] ARD においては、この SQLUINTEGER ヘッダー・フィールドには行セットの行数が指定されます。これは `SQLFetch()`、`SQLFetchScroll()`、または `SQLSetPos()` への呼び出しにより返される行の数です。デフォルト値は 1 です。 SQL_ATTR_ROW_ARRAY_SIZE 属性を指定した `SQLSetStmtAttr()` を呼び出してこのフィールドを設定することもできます。

APD においては、この SQLUINTEGER ヘッダー・フィールドには各パラメーターごとの値の数が指定されます。

このフィールドのデフォルト値は 1 です。 SQL_DESC_ARRAY_SIZE が 1 よりも大きい場合は、APD または ARD の SQL_DESC_DATA_PTR、SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR は配列を指しています。各配列のカーディナリティーは、このフィールドの値と同等です。

ARD 内のこのフィールドは、 SQL_ROWSET_SIZE 属性とともに `SQLSetStmtAttr()` を呼び出すことによっても設定できます。 APD 内のこのフィールドは、 SQL_ATTR_PARAMSET_SIZE 属性とともに `SQLSetStmtAttr()` を呼び出すことによっても設定できます。

SQL_DESC_ARRAY_STATUS_PTR [すべて] この SQLUSMALLINT * ヘッダー・フィールドは記述子タイプごとに、 SQLUSMALLINT 値の配列を指しています。この配列は以下のように呼ばれます。

- 行状況配列 (IRD)
- パラメーター状況配列 (IPD)
- 行操作配列 (ARD)
- パラメーター操作配列 (APD)

IRD においては、このヘッダー・フィールドは SQLFetch()、SQLFetchScroll()、または SQLSetPos() への呼び出しの後の状況値が入っている、行状況配列を指します。この配列のエレメントの数は、行セット内にある行の数と同じです。アプリケーションは SQLUSMALLINT の配列を割り当てて、このフィールドがその配列を指すように設定しなければなりません。デフォルト設定では、このフィールドは NULL ポインターに設定されます。 SQL_DESC_ARRAY_STATUS_PTR フィールドが NULL ポインターに設定されていない限り、 CLI は配列を移植しますが、これが行われるのは、状況値が生成されておらず、配列が移植されていない場合です。

注: 指し示される行状況配列のエレメントを、アプリケーションが IRD の SQL_DESC_ARRAY_STATUS_PTR フィールドによって設定する場合の動作は定義されていません。その配列は最初、SQLFetch()、SQLFetchScroll()、または SQLSetPos() への呼び出しによって移植されます。この呼び出しが SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合は、該当するフィールドによって指し示されているその配列の内容は定義されていません。

配列内のエレメントには、以下の値を入れることができます。

- SQL_ROW_SUCCESS: 行は正常にフェッチされ、最後のフェッチ以降は変更されていません。
- SQL_ROW_SUCCESS_WITH_INFO: 行は正常にフェッチされ、最後のフェッチ以降は変更されていません。しかし、行に関する警告が返されました。
- SQL_ROW_ERROR: その行のフェッチ中にエラーが生じました。
- SQL_ROW_UPDATED: 行は正常にフェッチされ、最後のフェッチ以降に更新されています。その行がもう一度フェッチされると、状況は SQL_ROW_SUCCESS となります。
- SQL_ROW_DELETED: 最後のフェッチ以降にその行は削除されています。
- SQL_ROW_ADDED: その行は SQLSetPos() により挿入されました。その行がもう一度フェッチされると、状況は SQL_ROW_SUCCESS となります。
- SQL_ROW_NOROW: 行セットが結果セットの最後とオーバーラップして、行状況配列のこのエレメントに対応していた行が 1 つも返されませんでした。

ARD 内のこのフィールドは、 SQL_ATTR_ROW_STATUS_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

IPD においては、このヘッダー・フィールドは SQLExecute() または SQLExecDirect() への呼び出し後、パラメーター値のセットごとの状況情報が入っているパラメーター状況配列を指しています。 SQLExecute() または SQLExecDirect() への呼び出しが SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合は、該当するフィールドによって指し示されているその配列の内容は定義されていません。アプリケーションは SQLUSMALLINT の配列を割り当てて、このフィールドがその配列を指すように設定しなければなりません。 SQL_DESC_ARRAY_STATUS_PTR フィールドが NULL ポインターに設定されていない限り、ドライバーは配列を移植しますが、これが行われるのは、状況値が生成されておらず、配列が移植されていない場合です。

配列内のエレメントには、以下の値を入れることができます。

- SQL_PARAM_SUCCESS: SQL ステートメントは、このパラメーターのセットに対して正常に実行されました。

- SQL_PARAM_SUCCESS_WITH_INFO: SQL ステートメントは、このパラメーターのセットに対して正常に実行されました。ただし、診断データ構造体の中に警告情報があります。
- SQL_PARAM_ERROR: このパラメーターのセットの処理中にエラーが生じました。診断データ構造体の中に追加のエラー情報があります。
- SQL_PARAM_UNUSED: このパラメーター・セットは使用できませんでした。前のパラメーター・セットのいずれかでエラーが発生し、処理が打ち切られたことが原因とみられます。
- SQL_PARAM_DIAG_UNAVAILABLE: 診断情報が利用できません。一例として、CLI がパラメーターの配列を一体構造の単位として処理することにより、このレベルのエラー情報が生成されない場合があげられます。

APD 内のこのフィールドは、 SQL_ATTR_PARAM_STATUS_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

ARD においては、このヘッダー・フィールドは、対象となる行を SQLSetPos() 操作で無視するかどうかを示すためにアプリケーションが設定できる値の、行操作配列を指します。

配列内のエレメントには、以下の値を入れることができます。

- SQL_ROW_PROCEED: この行は、 SQLSetPos() を使用したバルク操作に組み込まれています。 (この設定は、その操作がこの行で生じることを保証するわけではありません。この行の IRD 行状況配列における状況が SQL_ROW_ERROR であれば、 CLI が該当する操作をその行で実行できない場合があります。)
- SQL_ROW_IGNORE: この行は、 SQLSetPos() を使用したバルク操作から除外されています。

この配列のエレメントが 1 つも設定されていないと、すべての行がバルク操作に組み込まれます。また、ARD の SQL_DESC_ARRAY_STATUS_PTR フィールド内にある値が NULL ポインターである場合は、すべての行がバルク操作に組み込まれます。その変換処理は、ポインターが有効な配列を指していて、配列のすべてのエレメントが SQL_ROW_PROCEED である場合と同じです。配列中のあるエレメントが SQL_ROW_IGNORE に設定されていると、無視される行に対する行状況配列の値は変更されません。

ARD 内のこのフィールドは、 SQL_ATTR_ROW_OPERATION_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

APD においては、このヘッダー・フィールドは、該当するパラメーターのセットが SQLExecute() または SQLExecDirect() の呼び出し時に無視されるかどうかを示すためにアプリケーションが設定できる、値のパラメーター操作配列を指しています。配列内のエレメントには、以下の値を入れることができます。

- SQL_PARAM_PROCEED: パラメーターのセットが SQLExecute() または SQLExecDirect() 呼び出しに組み込まれています。
- SQL_PARAM_IGNORE: パラメーターのセットが SQLExecute() または SQLExecDirect() 呼び出しから除外されています。

この配列のエレメントが 1 つも設定されていないと、配列内のすべてのパラメーターのセットが SQLExecute() または SQLExecDirect() 呼び出しで使用されます。ま

た、APD の SQL_DESC_ARRAY_STATUS_PTR フィールド内にある値が NULL ポインターである場合は、すべてのパラメーターのセットが使用されます。その変換処理は、ポインターが有効な配列を指していて、配列のすべてのエレメントが SQL_PARAM_PROCEED である場合と同じです。

APD 内のこのフィールドは、 SQL_ATTR_PARAM_OPERATION_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

SQL_DESC_BIND_OFFSET_PTR [アプリケーション記述子] この SQLINTEGER * ヘッダー・フィールドは、バインド・オフセットを指します。デフォルト設定では、これは NULL ポインターに設定されます。このフィールドが NULL ポインターではない場合、CLI はそのポインターを参照解除して、フェッチ時の記述子レコードの中に NULL 以外の値が入っている据え置きフィールド (SQL_DESC_DATA_PTR、 SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR) のそれぞれに、その参照解除された値を追加します。そして、この新しいポインター値をバインド時に使用します。

バインド・オフセットは常に、SQL_DESC_DATA_PTR、SQL_DESC_INDICATOR_PTR、および SQL_DESC_OCTET_LENGTH_PTR フィールド内の値へ直接追加されます。そのオフセットが別の値に変更されると、この新しい値はそのまま各記述子フィールドの値に直接追加されます。新しいオフセットはフィールド値だけでなく、それ以前のどのオフセットにも追加されません。

このフィールドは*据え置きフィールド* です。これは設定時には使用されませんが、後で CLI によりデータを検索するのに使用されます。

ARD 内のこのフィールドは、 SQL_ATTR_ROW_BIND_OFFSET_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。 ARD 内のこのフィールドは、 SQL_ATTR_PARAM_BIND_OFFSET_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

SQL_DESC_BIND_TYPE [アプリケーション記述子] この SQLINTEGER ヘッダー・フィールドは、バインド方向を列またはパラメーターのいずれかのバインドに使用するよう設定します。

ARD においては、このフィールドは関連しているステートメント・ハンドルに対する SQLFetchScroll() の呼び出し時に、バインド方向を指定します。

列に対して列方向のバインドを選択するには、このフィールドを SQL_BIND_BY_COLUMN (デフォルト値) に設定します。

ARD 内のこのフィールドは、 SQL_ATTR_ROW_BIND_TYPE 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

APD においては、このフィールドは動的パラメーターに対して使用するバインド方向を指定します。

パラメーターに対して列方向のバインドを選択するには、このフィールドを SQL_BIND_BY_COLUMN (デフォルト値) に設定します。

APD 内のこのフィールドは、 SQL_ATTR_PARAM_BIND_TYPE 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

SQL_DESC_COUNT [すべて] この SQLSMALLINT ヘッダー・フィールドは、データが入っているレコードのうち最も番号の大きいレコードの 1 を基準とした指標を指定します。 CLI は、記述子に対してデータ構造を設定するときに、どれだけの数のレコードが有効であるかを示すよう COUNT フィールドも設定しなければなりません。アプリケーションがこのデータ構造のインスタンスを割り当てるときには、どれだけの数のレコードのために場所を予約しておくかをそのアプリケーションが指定する必要はありません。アプリケーションがレコードの内容を指定すると、記述子ハンドルに適切なサイズのデータ構造を確実に参照させるために必要な処置を CLI が行います。

SQL_DESC_COUNT は、バインドされるすべてのデータ列 (フィールドが ARD にある場合) またはバインドされるすべてのパラメーター (フィールドが APD にある場合) のカウントではなく、レコードのうち最も番号が大きいレコードの番号です。最も番号が大きい列の番号よりも小さい番号が付いた列またはパラメーターが (NULL ポインターに設定された *Target ValuePtr* 引数のある SQLBindCol()、または NULL ポインターに設定された *Parameter ValuePtr* 引数のある SQLBindParameter() を呼び出すことにより) アンバインドされている場合には、SQL_DESC_COUNT は変更されません。追加の列やパラメーターが、データが入っているレコードのうち最も番号が大きいレコードよりも大きな番号でバインドされた場合は、 CLI が自動的に SQL_DESC_COUNT フィールドにある値を増やします。すべての列またはパラメーターが、 SQL_UNBIND オプションを指定した SQLFreeStmt() の呼び出しによりアンバインドされた場合には、SQL_DESC_COUNT は 0 に設定されます。

SQL_DESC_COUNT の値は、アプリケーションが SQLSetDescField() を呼び出すことにより明示的に設定することができます。 SQL_DESC_COUNT の値を明示的に減少させると、 SQL_DESC_COUNT の新しい値より大きい番号のレコードはすべて除去され、その列はアンバインドされます。 SQL_DESC_COUNT の値が明示的に 0 に設定されると、そのフィールドが APD になっている場合は、すべてのパラメーターがアンバインドされます。 SQL_DESC_COUNT の値が明示的に 0 に設定されると、そのフィールドが ARD になっている場合は、バインド済みブックマーク列以外のすべてのデータ・バッファーが解放されます。

ARD のこのフィールド内にあるレコード・カウントには、バインド済みブックマーク列は含まれません。

SQL_DESC_ROWS_PROCESSED_PTR [インプリメンテーション記述子] IRD においては、この SQLUINTEGER * ヘッダー・フィールドは SQLFetch() または SQLFetchScroll() への呼び出しの後にフェッチされた行の数が入っている、あるいは SQLSetPos() への呼び出しによって実行されるバルク操作で影響を受ける行の数が入っているバッファーを指します。

IPD においては、この SQLUINTEGER * ヘッダー・フィールドはパラメーターの各行の処理時における行の番号が入っているバッファーを指しています。これが NULL ポインターであれば、行番号は返されません。

SQL_DESC_ROWS_PROCESSED_PTR が有効であるのは、 SQLFetch() または SQLFetchScroll() (IRD フィールドの場合)、もしくは SQLExecute() または SQLExecDirect() (IPD フィールドの場合) への呼び出しの後に、 SQL_SUCCESS または SQL_SUCCESS_WITH_INFO が返されている場合だけです。戻りコードが SQL_SUCCESS か SQL_SUCCESS_WITH_INFO でない場合は、SQL_DESC_ROWS_PROCESSED_PTR が指しているロケーションは未定義です。このフィールドが指しているバッファーに入っている呼び出しが SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合、バッファーの内容は SQL_NO_DATA が返されない限りは未定義であり、その場合にはバッファー内の値は 0 に設定されます。

ARD 内のこのフィールドは、 SQL_ATTR_ROWS_FETCHED_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。 ARD 内のこのフィールドは、 SQL_ATTR_PARAMS_PROCESSED_PTR 属性とともに SQLSetStmtAttr() を呼び出すことによっても設定できます。

このフィールドが指しているバッファーは、アプリケーションによって割り当てられます。これは、CLI により設定される据え置き出力バッファーです。デフォルト設定では、これは NULL ポインターに設定されます。

レコード・フィールド

各記述子には 1 つ以上のレコードが含まれており、それらのレコードは記述子のタイプによって列データまたは動的パラメーターのいずれかを定義するフィールドから構成されています。また各レコードは、単一の列またはパラメーターを完全に定義したものです。

SQL_DESC_AUTO_UNIQUE_VALUE [IRD] この読み取り専用の SQLINTEGER レコード・フィールドには、列が自動増分列である場合には SQL_TRUE が、または列が自動増分列でない場合には SQL_FALSE が入ります。このフィールドは読み取り専用ですが、基礎になっている自動増分列は必ずしも読み取り専用ではありません。

SQL_DESC_BASE_COLUMN_NAME [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、結果セット列のための基本列名が入っています。基本列名が存在しない場合 (列が式になっている場合など) は、この変数には空ストリングが入ります。

SQL_DESC_BASE_TABLE_NAME [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、結果セット列のための基本表名が入っています。基本表名が定義できないか適用外である場合、この変数には空ストリングが入ります。

SQL_DESC_CASE_SENSITIVE [インプリメンテーション記述子] この読み取り専用の SQLINTEGER レコード・フィールドには、照合または比較において列またはパラメーターがケース・センシティブとして扱われる場合には SQL_TRUE が、あるいは照合または比較において列がケース・センシティブとして扱われない場合や文字以外の列である場合には SQL_FALSE がそれぞれ入れられます。

SQL_DESC_CATALOG_NAME [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、該当する列が入れられる基本表のカタログ名または修飾子名が入っています。戻り値は、その列が式であるかビューの一部である場合には、ドライ

バーによって異なります。データ・ソースがカタログ (または修飾子) をサポートしていないか、カタログ名または修飾子名が判別できない場合は、この変数には空ストリングが入れられます。

SQL_DESC_CONCISE_TYPE [すべて] この SQLSMALLINT ヘッダー・フィールドは、すべてのデータ・タイプに対するコンサイス・データ・タイプを指定します。

SQL_DESC_CONCISE_TYPE および SQL_DESC_TYPE フィールドの値は相互に依存しています。一方のフィールドを設定するたびに、もう一方のフィールドも設定しなければなりません。 SQL_DESC_CONCISE_TYPE は、SQLBindCol() または SQLBindParameter()、あるいは SQLSetDescField() への呼び出しで設定できます。SQL_DESC_TYPE は、 SQLSetDescField() または SQLSetDescRec() への呼び出しで設定できます。

SQL_DESC_CONCISE_TYPE をコンサイス・データ・タイプに設定すると、SQL_DESC_TYPE フィールドはそれと同じ値に設定され、SQL_DESC_DATETIME_INTERVAL_CODE フィールドは 0 に設定されます。

SQL_DESC_DATA_PTR [アプリケーション記述子および IPD] この SQLPOINTER レコード・フィールドは、パラメーター値 (APD の場合) または列値 (ARD の場合) が入れられる変数を指します。記述子レコード (および、それが表す列またはパラメーターのいずれか) は、 SQLBindCol() または SQLBindParameter() のいずれかへの呼び出し内の Target*ValuePtr* が NULL ポインターであるか、SQLSetDescField() または SQLSetDescRec() への呼び出し内の SQL_DESC_DATA_PTR フィールドが NULL ポインターに設定されると、アンバインドされます。 SQL_DESC_DATA_PTR フィールドが NULL ポインターに設定されていれば他のフィールドは影響されません。このフィールドが指しているバッファーに入っている SQLFetch() または SQLFetchScroll() が SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合、そのバッファーの内容は未定義です。

このフィールドは据え置きフィールドです。これは設定時には使用されませんが、後で CLI によりデータを検索するのに使用されます。

SQL_DESC_DATA_PTR フィールドが設定されると、 CLI はいつでも SQL_DESC_TYPE フィールド内の値に含まれている CLI または ODBC データ・タイプが正しいかどうかと、そのデータ・タイプに影響する他のすべてのフィールドの整合性が保たれているかをチェックします。詳細は、整合性チェックの解説の項を参照してください。

SQL_DESC_DATETIME_INTERVAL_CODE [すべて] この SQLSMALLINT レコード・フィールドには、 SQL_DESC_TYPE フィールドが SQL_DATETIME である場合の、特定の日時データ・タイプに対するサブコードが入っています。これは、SQL および C の両方のデータ・タイプに当てはまります。

このフィールドは、日時データ・タイプに対して以下のように設定できます。

表 169. 日時サブコード

日時のタイプ	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME/SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIMESTAMP/ SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

ODBC 3.0 では、CLI がサポートしていないインターバル用の他の値 (ここには示されていません) が定義されています。 SQLSetDescRec() または SQLSetDescField() の呼び出しで他の値を指定した場合はすべて、 HY092 (オプション・タイプが範囲外です) のエラーが生じます。

SQL_DESC_DATETIME_INTERVAL_PRECISION [すべて] ODBC 3.0 はこの SQLINTEGER レコード・フィールドを定義していますが、CLI はインターバル・データ・タイプをサポートしていません。戻される固定値は 0 です。このフィールドを設定しようとすると、 01S02 (オプション値が変更されました) が出されます。

SQL_DESC_DISPLAY_SIZE [IRD] この読み取り専用の SQLINTEGER レコード・フィールドには、列からのデータを表示するのに必要な最大文字数が入っています。このフィールド内の値は記述子フィールド SQL_DESC_LENGTH と同じではありません。それは、この LENGTH フィールドはすべての数値タイプに対して未定義であるためです。

SQL_DESC_FIXED_PREC_SCALE [インプリメンテーション記述子] この読み取り専用の SQLSMALLINT レコード・フィールドは、列が厳密な数列であり、精度が固定されていてスケールがゼロ以外である場合には SQL_TRUE に、また列が厳密な数列ではなく、精度とスケールが固定されていない場合には SQL_FALSE に設定されます。

SQL_DESC_INDICATOR_PTR [アプリケーション記述子] ARD においては、この SQLINTEGER * レコード・フィールドは標識変数を指します。この変数には、列値が NULL である場合は SQL_NULL_DATA が入れられます。 APD の場合、この標識変数は SQL_NULL_DATA に設定され、NULL 動的引数が指定されます。それ以外の場合には、この変数はゼロです (SQL_DESC_INDICATOR_PTR および SQL_DESC_OCTET_LENGTH_PTR の値が同じポインターである場合を除く)。

ARD 内の SQL_DESC_INDICATOR_PTR フィールドが NULL ポインターである場合、 CLI は列が NULL であるかどうかの情報を返すことができなくなります。列が NULL であり、INDICATOR_PTR が NULL ポインターである場合は、 CLI が SQLFetch() または SQLFetchScroll() への呼び出し後にバッファーを移植しようとする時点で、 SQLSTATE 22002、「標識変数が必要だが指定されていない (Indicator variable required but not supplied)」が返されます。 SQLFetch() または SQLFetchScroll() への呼び出しが SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合、バッファーの内容は定義されていません。

SQL_DESC_INDICATOR_PTR フィールドは、 SQL_DESC_OCTET_LENGTH_PTR が指しているフィールドが設定されているかどうかを判別します。ある列のデータ値が NULL になっていると、CLI はその標識変数を SQL_NULL_DATA に設定し

ます。 SQL_DESC_OCTET_LENGTH_PTR が指しているフィールドはその時点では設定されません。フェッチの途中で NULL 値が検出されなければ、SQL_DESC_INDICATOR_PTR が指しているバッファーはゼロに設定され、SQL_DESC_OCTET_LENGTH_PTR が指しているバッファーはデータの長さに設定されます。

APD 内にある INDICATOR_PTR フィールドが NULL ポインターである場合、アプリケーションはこの記述子レコードを使って NULL 引数を指定することができません。

このフィールドは据え置きフィールドです。これは設定時には使用されませんが、後で CLI によりデータを格納するのに使用されます。

SQL_DESC_LABEL [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、列ラベルまたは列タイトルが入っています。列にラベルがない場合、この変数には列名が入れられます。列に名前やラベルが付けられていないと、この変数には空ストリングが入れられます。

SQL_DESC_LENGTH [すべて] この SQLUINTEGER レコード・フィールドは、文字ストリングの最大もしくは実際の文字長か、あるいはバイナリー・データ・タイプです。これは、固定長データ・タイプの場合は最大文字長、可変長データ・タイプの場合は実際の文字長となります。その値からは常に、文字ストリングの終わりを示す NULL 終止符文字が除かれています。このフィールドは文字数を示しており、バイト数を示しているのではない点に注意してください。

SQL_DESC_LITERAL_PREFIX [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、CLI がこのデータ・タイプのリテラルにおける接頭部として認識する 1 つ以上の文字が入っています。リテラルの接頭部が適用外であるデータ・タイプに対しては、この変数に空ストリングが入れられます。

SQL_DESC_LITERAL_SUFFIX [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、CLI がこのデータ・タイプのリテラルにおける接尾部として認識する 1 つ以上の文字が入っています。リテラルの接尾部が適用外であるデータ・タイプに対しては、この変数に空ストリングが入れられます。

SQL_DESC_LOCAL_TYPE_NAME [インプリメンテーション記述子] この読み取り専用の SQLCHAR レコード・フィールドには、データ・タイプのローカライズされた (ネイティブ言語の) 名前が入れられます。この名前は、データ・タイプの正規名とは異なることがあります。ローカライズされた名前がない場合は、空ストリングが返されます。このフィールドは、表示の目的においてのみ使用されます。

SQL_DESC_NAME [インプリメンテーション記述子] 行の記述子内にあるこの SQLCHAR レコード・フィールドには、列の別名が入れられます (該当する場合)。列の別名が適用されない場合には、列名が返されます。いずれの場合でも、UNNAMED フィールドは SQL_NAMED に設定されます。列名も列の別名もなければ、NAME フィールドには空ストリングが返され、 UNNAMED フィールドは SQL_UNNAMED に設定されます。

アプリケーションは IPD の SQL_DESC_NAME フィールドをパラメーター名やその別名に設定して、ストアード・プロシージャーのパラメーターを名前で指定することができます。 IRD の SQL_DESC_NAME フィールドは読み取り専用フィール

ドであるため、アプリケーションがそのフィールドを設定しようとすると SQLSTATE HY091 (記述子のフィールド ID が無効 (Invalid descriptor field identifier)) が返されます。

IPD においては、このフィールドは動的パラメーターがサポートされていない場合、未定義になります。名前付きパラメーターがサポートされており、そのバージョンの CLI でパラメーターの記述ができるようになっている場合は、このフィールドにはパラメーター名が返されます。

列名値は、SQLSetEnvAttr() で設定された環境属性 SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA の影響を受ける場合があります。

SQL_DESC_NULLABLE [インプリメンテーション記述子] IRD では、この読み取り専用の SQLSMALLINT レコード・フィールドは、列に NULL 値を入れることができる場合には SQL_NULLABLE、列に NULL 値を入れられない場合には SQL_NO_NULLS、そして列に NULL 値を入れることができるかどうか不明である場合には SQL_NULLABLE_UNKNOWN となります。このフィールドは基本列ではなく、結果セット列に属しています。

IPD においては、動的パラメーターが常に NULL 可能であるため、このフィールドは常に SQL_NULLABLE に設定され、アプリケーションはこれを設定することができません。

SQL_DESC_NUM_PREC_RADIX [すべて] この SQLINTEGER フィールドには、SQL_DESC_TYPE フィールド内のデータ・タイプが近似値データ・タイプである場合は値として 2 が入れられます。これは、SQL_DESC_PRECISION フィールドに入っているのがビット数であるためです。また、 SQL_DESC_TYPE フィールド内のデータ・タイプが厳密な数データ・タイプである場合は値として 10 が入れられます。これは、SQL_DESC_PRECISION フィールドに入っているのが小数桁数であるためです。数値以外のすべてのデータ・タイプに対しては、このフィールドは 0 に設定されます。

SQL_DESC_OCTET_LENGTH [すべて] この SQLINTEGER レコード・フィールドには、文字ストリング・データ・タイプまたはバイナリー・データ・タイプの長さがバイト単位で入れられます。固定長文字タイプの場合、これは実際の長さ (バイト単位) です。可変長文字タイプまたはバイナリー・タイプの場合、これは最大長 (バイト単位) になります。この値は常に、インプリメンテーション記述子の場合には NULL 終止符文字のためのスペースを除外してあり、アプリケーション記述子の場合には NULL 終止符文字のためのスペースが含められています。アプリケーション・データの場合、このフィールドにはそのバッファーのサイズが入れられます。APD の場合、このフィールドは出力パラメーターまたは入出力パラメーターに対してのみ定義されます。

SQL_DESC_OCTET_LENGTH_PTR [アプリケーション記述子] この SQLINTEGER * レコード・フィールドが指している変数には、動的引数 (パラメーター記述子の場合) の、あるいはバインド済み列値 (行記述子の場合) の合計長がバイト単位で入れられます。

APD の場合、文字ストリングとバイナリー数を除くすべての引数において無視されます。このフィールドが SQL_NTS を指しているなら、その動的引数はヌル終了で

なければなりません。バインド済みパラメーターを実行時データ・パラメーターにすることを示すため、アプリケーションは APD の適切なレコード内にあるこのフィールドを、実行時に値 SQL_DATA_AT_EXEC が入れられる変数に設定します。これと同様のフィールドが複数ある場合には、該当するパラメーターを固有に識別できるような値に SQL_DESC_DATA_PTR を設定することができ、これによってアプリケーションは要求されているパラメーターがどれであるかを判別しやすくなります。

ARD の OCTET_LENGTH_PTR フィールドが NULL ポインターである場合、 CLI はその列の長さ情報を返しません。また、APD の SQL_DESC_OCTET_LENGTH_PTR フィールドが NULL ポインターである場合は、CLI は文字ストリングとバイナリー値がヌル終了であるとみなします。 (バイナリー値はヌル終了であってはなりませんが、切り捨てが生じないよう長さが指定されていなければなりません。)

このフィールドが指しているバッファーに入っている SQLFetch() または SQLFetchScroll() が SQL_SUCCESS または SQL_SUCCESS_WITH_INFO を返さなかった場合、そのバッファーの内容は未定義です。

このフィールドは据え置きフィールドです。これは設定時には使用されませんが、後で CLI によりデータをバッファーに入れるのに使用されます。

デフォルト設定では、これは 4 バイト値へのポインターです。

SQL_DESC_PARAMETER_TYPE [IPD] この SQLSMALLINT レコード・フィールドは、入力パラメーターの場合には SQL_PARAM_INPUT に、入出力パラメーターの場合には SQL_PARAM_INPUT_OUTPUT に、また出力パラメーターの場合には SQL_PARAM_OUTPUT に設定されます。デフォルト設定では、SQL_PARAM_INPUT に設定されます。

IPD の場合、IPD が CLI により自動的に移植されないと、このフィールドはデフォルトで SQL_PARAM_INPUT に設定されます (SQL_ATTR_ENABLE_AUTO_IPD ステートメント属性は SQL_FALSE です)。アプリケーションは IPD におけるこのフィールドを、入力パラメーターではないパラメーターに対して設定すべきです。

SQL_DESC_PRECISION [すべて] この SQLSMALLINT レコード・フィールドには、厳密な数タイプの場合には桁数が、近似値タイプの場合には仮数 (バイナリー精度) におけるビット数が、また SQL_TYPE_TIME または SQL_TYPE_TIMESTAMP のタイプの場合には秒の部分の小数桁数が入れられます。それ以外のすべてのデータ・タイプに対しては、このフィールドは未定義となります。

SQL_DESC_SCALE [すべて] この SQLSMALLINT レコード・フィールドには、DECIMAL および NUMERIC データ・タイプの場合には定義済みのスケールが入れられます。それ以外のすべてのデータ・タイプに対しては、このフィールドは未定義となります。

SQL_DESC_SCHEMA_NAME [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、対象となる列が入っている基本表のスキーマ名が入れられます。多

くの DBMS の場合、これは所有者名となります。データ・ソースがスキーマ (または所有者) をサポートしていないか、スキーマ名が判別できない場合は、この変数には空ストリングが入れられます。

SQL_DESC_SEARCHABLE [IRD] この読み取り専用の SQLSMALLINT レコード・フィールドは、以下のいずれかの値に設定されます。

- 列を WHERE 節で使用できない場合は、SQL_PRED_NONE。 (これは、ODBC 2.0 で定義されている SQL_UNSEARCHABLE 値と同じです。)
- 列を WHERE 節で使用できるが、LIKE 述部を指定したときに限られる場合は、SQL_PRED_CHAR。 (これは、ODBC 2.0 で定義されている SQL_LIKE_ONLY 値と同じです。)
- LIKE 以外のすべての比較演算子を指定した WHERE 節で列を使用できる場合は、SQL_PRED_BASIC。 (これは、ODBC 2.0 で定義されている SQL_EXCEPT_LIKE 値と同じです。)
- 任意の比較演算子を指定した WHERE 節で列を使用できる場合は、SQL_PRED_SEARCHABLE。

SQL_DESC_TABLE_NAME [IRD] この読み取り専用の SQLCHAR レコード・フィールドには、対象となる列が入っている基本表の名前が入れられます。

SQL_DESC_TYPE [すべて] この SQLSMALLINT レコード・フィールドは、すべてのデータ・タイプに対する SQL または C のコンサイス・データ・タイプを指定します。

注: ODBC 3.0 は、CLI によってサポートされていない SQL_INTERVAL データ・タイプを定義しています。このデータ・タイプに関連したどの動作も CLI 内では存在しません。

SQL_DESC_TYPE および SQL_DESC_CONCISE_TYPE フィールドの値は相互に依存しています。一方のフィールドを設定するたびに、もう一方のフィールドも設定しなければなりません。 SQL_DESC_TYPE は、 `SQLSetDescField()` または `SQLSetDescRec()` への呼び出しで設定できます。 SQL_DESC_CONCISE_TYPE は、`SQLBindCol()` または `SQLBindParameter()`、あるいは `SQLSetDescField()` への呼び出しで設定できます。

SQL_DESC_TYPE をコンサイス・データ・タイプに設定すると、SQL_DESC_CONCISE_TYPE フィールドはそれと同じ値に設定され、SQL_DESC_DATETIME_INTERVAL_CODE フィールドは 0 に設定されます。

`SQLSetDescField()` を呼び出して SQL_DESC_TYPE フィールドを設定すると、以下のフィールドが次のようなデフォルト値に設定されます。なお、同じレコードの残りのフィールドの値は未定義です。

表 170. デフォルト値

SQL_DESC_TYPE	暗黙的に設定される他のフィールド
SQL_CHAR、SQL_VARCHAR	SQL_DESC_LENGTH は 1 に設定されます。 SQL_DESC_PRECISION は 0 に設定されます。

表 170. デフォルト値 (続き)

SQL_DESC_TYPE	暗黙的に設定される他のフィールド
SQL_DECIMAL、SQL_NUMERIC	SQL_DESC_SCALE は 0 に設定されます。SQL_DESC_PRECISION は、データ・タイプの精度に設定されます。
SQL_FLOAT	SQL_DESC_PRECISION は、 SQL_FLOAT のデフォルトの精度に設定されます。
SQL_DATETIME	SQL_DESC_CONCISE_TYPE と SQL_DESC_DATETIME_INTERVAL_CODE の片方または両方が暗黙で設定されて、DATE SQL タイプまたは C タイプを指定することがあります。
SQL_INTERVAL	このデータ・タイプは CLI ではサポートされません。

アプリケーションが SQLSetDescRec() を呼び出す代わりに SQLSetDescField() を呼び出して記述子のフィールドを設定するときは、そのアプリケーションは最初にデータ・タイプを宣言しなければなりません。暗黙的に設定された値が受諾不能であれば、アプリケーションは次に SQLSetDescField() を呼び出してその受諾不能の値を明示的に設定できます。

SQL_DESC_TYPE_NAME [インプリメンテーション記述子] この読み取り専用の SQLCHAR レコード・フィールドには、データ・ソースに依存するタイプの名前 (例えば、CHAR、 VARCHAR など) が入れられます。該当するデータ・タイプの名前が不明である場合は、この変数には空ストリングが入れられます。

SQL_DESC_UNNAMED [インプリメンテーション記述子] 行記述子の中にあるこの SQLSMALLINT レコード・フィールドは、 SQL_NAMED または SQL_UNNAMED のいずれかに設定されます。 NAME フィールドに列の別名が入っている場合、あるいは列の別名を適用しない場合は、 UNNAMED フィールドは SQL_NAMED に設定されます。また、列名や列の別名がない場合には、UNNAMED フィールドが SQL_UNNAMED に設定されます。

アプリケーションは IPD の SQL_DESC_UNNAMED フィールドを SQL_UNNAMED に設定することができます。アプリケーションが IPD の SQL_DESC_UNNAMED フィールドを SQL_NAMED に設定しようとすると、SQLSTATE HY091 (記述子のフィールド ID が無効 (Invalid descriptor field identifier)) が返されます。 IRD の SQL_DESC_UNNAMED フィールドは読み取り専用であるため、アプリケーションがそのフィールドを設定しようとすると SQLSTATE HY091 (記述子のフィールド ID が無効 (Invalid descriptor field identifier)) が返されます。

SQL_DESC_UNSIGNED [インプリメンテーション記述子] この読み取り専用の SQLSMALLINT レコード・フィールドは、列タイプが無符号または非数値の場合には SQL_TRUE に、列タイプが符号ありの場合には SQL_FALSE に設定されます。

SQL_DESC_UPDATABLE [IRD] この読み取り専用の SQLSMALLINT レコード・フィールドは、以下のいずれかの値に設定されます。

- 結果セット列が読み取り専用の場合、SQL_ATTR_READONLY。
- 結果セット列が読み取り/書き込み指定の場合、SQL_ATTR_WRITE。
- 結果セット列が更新可能かどうかが不明である場合、SQL_ATTR_READWRITE_UNKNOWN。

SQL_DESC_UPDATABLE は、基本表内の列ではなく、結果セット内の列の更新可能度を記述します。この結果セット列の元になっている基本表内の列の更新可能度は、このフィールド内の値とは異なっていることがあります。また、列が更新可能であるかどうかは、データ・タイプ、ユーザー特権、および結果セットそのものの定義に基づいていることが考えられます。列が更新可能であるかどうかが不明であれば、 SQL_UPDT_READWRITE_UNKNOWN が返されることになります。

SQL_DESC_USER_DEFINED_TYPE_CODE [IRD] これは、読み取り専用の SQLINTEGER であり、列のデータ・タイプの特性を説明する情報を戻します。戻される値は以下の 4 つのいずれかです。

- SQL_TYPE_BASE: 列のデータ・タイプは、基本のデータ・タイプ (CHAR、DATE、または DOUBLE など) です。
- SQL_TYPE_DISTINCT: 列のデータ・タイプは、ユーザー定義の別個のタイプです。
- SQL_TYPE_REFERENCE: 列のデータ・タイプは、参照用のユーザー定義タイプです。
- SQL_TYPE_STRUCTURED: 列のデータ・タイプは、構造化されたユーザー定義タイプです。

SQL_DESC_CARDINALITY [APD, IPD] この SQLLEN レコード・フィールドは、指定されたパラメーター・マーカーの配列値の最大カーディナリティーを示します。IPD 値は、配列値用にデータベースに送信できる最大カーディナリティーを示します。APD 値は、配列値に対する最大カーディナリティーですが、アプリケーションはこの方法によって、CALL ステートメントに対する出力パラメーター配列値に割り振られるストレージの最大容量を示します。

SQL_DESC_CARDINALITY_PTR [APD] この SQLLEN * レコード・フィールドは、ステートメントの実行時に、パラメーターのカーディナリティーを含む変数を指します。入力パラメーター・マーカーの場合、アプリケーションはこの方法によって、配列値の実際のカーディナリティーを提供します。出力パラメーター・マーカーの場合、これは戻される配列値のカーディナリティーを CLI が示す場所です。出力パラメーターの場合、この値は、ストアード・プロシージャーによって戻される配列のカーディナリティーを示すことに注意してください。このストアード・プロシージャーから戻される実際のカーディナリティーよりも SQL_DESC_CARDINALITY(APD) が小さい可能性があるため、カーディナリティーはアプリケーションに必ずしも書き込まれるわけではありません。これは据え置きフィールドです。これは設定時には使用されませんが、後で CLI に使用されます。

記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)

以下の表は、各記述子タイプごとの各フィールドの初期設定を示しています。なお D は、そのフィールドがデフォルト値を使って初期設定され、 ND は、そのフィールドがデフォルト値を使わないで初期設定されることを意味します。数字が示されている場合は、その数字がフィールドのデフォルト値です。この表にはまた、フィールドが読み取り/書き込み (R/W) であるか、読み取り専用 (R) であるかも示されています。

ヘッダー・フィールドの初期設定は次のとおりです。

表 171. ヘッダー・フィールドの初期設定

記述子ヘッダー・フィールド	タイプ	読み取り/書き込み (R/W) または読み取り専用 (R)	初期設定値
SQL_DESC_ALLOC_TYPE	SQLSMALLINT	• ARD: R • APD: R • IRD: R • IPD: R	• ARD: SQL_DESC_ALLOC_AUTO (明示指定の場合) または SQL_DESC_ALLOC_USER (暗黙指定の場合) • APD: SQL_DESC_ALLOC_AUTO (明示指定の場合) または SQL_DESC_ALLOC_USER (暗黙指定の場合) • IRD: SQL_DESC_ALLOC_AUTO • IPD: SQL_DESC_ALLOC_AUTO
SQL_DESC_ARRAY_SIZE	SQLUINTEGER	• ARD: R/W • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: [a] • APD: [a] • IRD: 未使用 • IPD: 未使用
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT *	• ARD: R/W • APD: R/W • IRD: R/W • IPD: R/W	• ARD: NULL ポインター • APD: NULL ポインター • IRD: NULL ポインター • IPD: NULL ポインター
SQL_DESC_BIND_OFFSET_PTR	SQLINTEGER *	• ARD: R/W • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: NULL ポインター • APD: NULL ポインター • IRD: 未使用 • IPD: 未使用
SQL_DESC_BIND_TYPE	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: SQL_BIND_BY_COLUMN • APD: SQL_BIND_BY_COLUMN • IRD: 未使用 • IPD: 未使用
SQL_DESC_COUNT	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: 0 • APD: 0 • IRD: D • IPD: 0
SQL_DESC_ROWS_ PROCESSED_PTR	SQLUINTEGER *	• ARD: 未使用 • APD: 未使用 • IRD: R/W • IPD: R/W	• ARD: 未使用 • APD: 未使用 • IRD: NULL ポインター • IPD: NULL ポインター

a これらのフィールドは、IPD が CLI によって自動的に移植される場合にのみ定義されます。フィールドが自動的に移植されない場合には定義されません。アプリケーションがこれらのフィールドを設定しようとすると、SQLSTATE HY091 (記述子のフィールド ID が無効) が返されます。

レコード・フィールドの初期設定は以下のとおりです。

表 172. レコード・フィールドの初期設定

記述子レコード・フィールド	タイプ	読み取り/書き込み (R/W) または読み取り専用 (R)	初期設定値
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: D [a]
SQL_DESC_CATALOG_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: SQL_C_DEFAULT • APD: SQL_C_DEFAULT • IRD: D • IPD: ND
SQL_DESC_DATA_PTR	SQLPOINTER	• ARD: R/W • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: NULL ポインター • APD: NULL ポインター • IRD: 未使用 • IPD: 未使用 [b]
SQL_DESC_DATETIME_ INTERVAL_CODE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_DATETIME_ INTERVAL_PRECISION	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_DISPLAY_SIZE	SQLINTEGER	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: D [a]

表 172. レコード・フィールドの初期設定 (続き)

記述子レコード・フィールド	タイプ	読み取り/書き込み (R/W) または読み取り専用 (R)	初期設定値
SQL_DESC_INDICATOR_PTR	SQLINTEGER *	• ARD: R/W • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: NULL ポインター • APD: NULL ポインター • IRD: 未使用 • IPD: 未使用
SQL_DESC_LABEL	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_LENGTH	SQLUINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: D [a]
SQL_DESC_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_NULLABLE	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R	• ARD: ND • APD: ND • IRD: N • IPD: ND
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_OCTET_LENGTH	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_OCTET_LENGTH_PTR	SQLINTEGER *	• ARD: R/W • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: NULL ポインター • APD: NULL ポインター • IRD: 未使用 • IPD: 未使用

表 172. レコード・フィールドの初期設定 (続き)

記述子レコード・フィールド	タイプ	読み取り/書き込み (R/W) または読み取り専用 (R)	初期設定値
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IPD: 未使用 • IRD: R/W	• ARD: 未使用 • APD: 未使用 • IPD: 未使用 • IRD: D=SQL_PARAM_INPUT
SQL_DESC_PRECISION	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_SCALE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_SCHEMA_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_SEARCHABLE	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_TABLE_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用
SQL_DESC_TYPE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: SQL_C_DEFAULT • APD: SQL_C_DEFAULT • IRD: D • IPD: ND
SQL_DESC_TYPE_NAME	SQLCHAR *	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: D [a]
SQL_DESC_UNNAMED	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_UNSIGNED	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: R	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: D [a]
SQL_DESC_UPDATABLE	SQLSMALLINT	• ARD: 未使用 • APD: 未使用 • IRD: R • IPD: 未使用	• ARD: 未使用 • APD: 未使用 • IRD: D • IPD: 未使用

表 172. レコード・フィールドの初期設定 (続き)

記述子レコード・フィールド	タイプ	読み取り/書き込み (R/W) または読み取り専用 (R)	初期設定値
SQL_DESC_CARDINALITY	SQLLEN	• ARD: 未使用 • APD: R/W • IRD: 未使用 • IPD: R/W	• ARD: 未使用 • APD: D • IRD: 未使用 • IPD: D
SQL_DESC_CARDINALITY_PTR	SQLLEN *	• ARD: 未使用 • APD: R/W • IRD: 未使用 • IPD: 未使用	• ARD: 未使用 • APD: D • IRD: 未使用 • IPD: 未使用

a これらのフィールドは、IPD が CLI によって自動的に移植される場合にのみ定義されます。フィールドが自動的に移植されない場合には定義されません。アプリケーションがこれらのフィールドを設定しようとすると、SQLSTATE HY091 (記述子のフィールド ID が無効) が返されます。

b IPD 内の SQL_DESC_DATA_PTR フィールドは、整合性検査を強制的に行わせるように設定できます。その後の `SQLGetDescField()` または `SQLGetDescRec()` への呼び出しにおいて、 CLI は SQL_DESC_DATA_PTR が設定された値を返す必要はありません。

第 6 章 DiagIdentifier 引数 (CLI) のヘッダー・フィールドとレコード・フィールド

ヘッダー・フィールド

以下のヘッダー・フィールドを、*DiagIdentifier* 引数に指定することができます。記述子フィールドに定義できる診断ヘッダー・フィールドは、 SQL_DIAG_NUMBER および SQL_DIAG_RETURNCODE のみです。

表 173. DiagIdentifier 引数のヘッダー・フィールド

ヘッダー・フィールド	戻りのタイプ	説明
SQL_DIAG_CURSOR_ROW_COUNT	SQLINTEGER	カーソルにある行のカウントです。フィールドのセマンティクスは、SQLGetInfo() 情報タイプにより異なります。これらは、カーソル・タイプごとにどの行カウントが使用できるかを示します (SQL_CA2_CRC_EXACT および SQL_CA2_CRC_APPROXIMATE ビットにおいて)。 • SQL_DYNAMIC_CURSOR_ATTRIBUTES2 • SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2 • SQL_KEYSET_CURSOR_ATTRIBUTES2 • SQL_STATIC_CURSOR_ATTRIBUTES2 このフィールドでは、値はステートメント・ハンドルだけに、また SQLExecute() 、SQLExecDirect()、または SQLMoreResults() 関数への呼び出しの後だけに定義されます。 ステートメント・ハンドル以外のハンドルで、 SQL_DIAG_CURSOR_ROW_COUNT の *DiagIdentifier* 引数値を指定して SQLGetDiagField() 関数を呼び出すと、SQL_ERROR が返されます。

表 173. DiagIdentifier 引数のヘッダー・フィールド (続き)

ヘッダー・フィールド	戻りのタイプ	説明
SQL_DIAG_DYNAMIC_ FUNCTION	CHAR *	基礎となる関数が実行した SQL ステートメントを記述するストリングです (CLI がサポートしている値については、 動的関数フィールドの値を参照してください)。このフィールドでは、値はステートメント・ハンドルだけに、また SQLExecute()、SQLExecDirect()、または SQLMoreResults() 関数への呼び出しの後だけに定義されます。
SQL_DIAG_DYNAMIC_ FUNCTION_CODE	SQLINTEGER	基礎となる関数が実行した SQL ステートメントを記述する数字コードです (CLI がサポートしている値については、 動的関数フィールドの値を参照してください)。このフィールドでは、値はステートメント・ハンドルだけに、また SQLExecute()、SQLExecDirect()、または SQLMoreResults() 関数への呼び出しの後だけに定義されます。 ステートメント・ハンドル以外のハンドルで、SQL_DIAG_DYNAMIC_FUNCTION_CODE の *DiagIdentifier* 引数値を指定して SQLGetDiagField() 関数を呼び出すと、SQL_ERROR が返されます。

表 173. DiagIdentifier 引数のヘッダー・フィールド (続き)

ヘッダー・フィールド	戻りのタイプ	説明
SQL_DIAG_NETWORK_STATISTICS	SQL_NET_STATS **注:** SQL_NET_STATS 構造は `sqlcli1.h` ファイル内で定義されます。	接続のネットワーク統計が含まれている構造体です。 この統計には、以下の情報が含まれています。 • データベース処理時間 (マイクロ秒単位) • (データベース処理時間を含む) ネットワーク時間 (マイクロ秒単位) • データベース・サーバーに送信したバイト数 • データベース・サーバーから受信したバイト数 • DRDA ラウンドトリップ数 このフィールドは、バージョン 9.7 フィックスパック 3 以降で使用可能です。 CLI は、SQL_ATTR_NETWORK_STATISTICS 接続属性が有効な場合に接続に関する統計を集計します。アプリケーションが SQLGetDiagField() 関数を呼び出して統計を取得すると、CLI は、統計を集計するために接続が使用している内部カウンターをリセットします。このフィールドでは、値は接続ハンドル SQL_HANDLE_DBC だけに定義されます。 • *DiagIdentifier* 引数に SQL_DIAG_NETWORK_STATISTICS を指定し、SQL_HANDLE_ENV を指定して SQLGetDiagField() 関数を呼び出すと、ERROR が戻ります。 • *DiagIdentifier* 引数に SQL_DIAG_NETWORK_STATISTICS を指定し、SQL_HANDLE_STMT または SQL_HANDLE_DESC を指定して SQLGetDiagField() 関数を呼び出すと、指定のステートメントまたは記述子ハンドルに関連する基礎となる接続ハンドルの診断情報 ID フィールドが戻ります。
SQL_DIAG_NUMBER	SQLINTEGER	指定されたハンドルで使用できる状況レコードの数です。
SQL_DIAG_RELATIVE_ COST_ESTIMATE	SQLINTEGER	SQLPrepare() 関数が正常に呼び出された場合にステートメントを処理するために必要なリソースの、相対コスト見積もりです。据え置き準備が使用できる場合、このフィールドの値は、ステートメントが実行されるまで、0 になります。

表 173. DiagIdentifier 引数のヘッダー・フィールド (続き)

ヘッダー・フィールド	戻りのタイプ	説明
SQL_DIAG_RETURNCODE	RETCODE	指定されたハンドルに関連した、最後に実行された関数の戻りコードです。 *Handle* で関数が呼び出されていなければ、SQL_DIAG_RETURNCODE フィールドには SQL_SUCCESS が返されます。
SQL_DIAG_ROW_COUNT	SQLINTEGER	SQLExecute()、SQLExecDirect()、または SQLSetPos() 関数により実行される、挿入、削除、または更新で影響を受ける行の数です。このフィールドの値はカーソル指定が実行された後、ステートメント・ハンドルに対してのみ定義されます。フィールド内のデータは、SQLRowCount() 関数の *RowCountPtr* 引数に返されます。フィールド内のデータは毎回の関数呼び出し後にリセットされますが、SQLRowCount() 関数によって返される行カウントは、ステートメントの状態が準備済みまたは割り振り済みの状態にリセットされるまでは同じ状態に保たれます。
SQL_DIAG_TOLERATED_ERROR		WebSphere® Federated Server V9.1 から、Error Tolerant Nested Table Expression (ETNTE) を指定できるようになりました。これにより、`UNION ALL` 照会時に 1 つ以上のデータ・ソースが使用不可のときに許容できるエラーと特別に使用するエラーを指定できます。いずれかの共用体の欠如 (ソースがオフラインのため) を許容することで、結果セットの処理を続行でき、**`SQL_NO_DATA_FOUND`**(100) が返されるまで `SQLFetch` を繰り返し呼び出すことができます。 許容エラーが検出された場合、CLI アプリケーションは次の 2 つのいずれかを行ってそれを発見します。 • **`DiagIdentifier`** 引数値に SQL_DIAG_TOLERATED_ERROR を指定して `SQLGetDiagField()` 関数を呼び出す。許容エラーが処理された場合は、TRUE が返されます。それ以外の場合は、FALSE が返されます。 • `SQLGetDiagRec()` 関数を呼び出す。許容エラーが処理された場合は、診断レコードにエラー SQL20383W が示されます。それ以外の場合は、エラー SQL0100W が示されます。

レコード・フィールド

以下のレコード・フィールドを、*DiagIdentifier* 引数に指定することができます。

表 174. DiagIdentifier 引数のレコード・フィールド

レコード・フィールド	戻りのタイプ	説明
SQL_DIAG_CLASS_ORIGIN	CHAR *	レコード内の SQLSTATE 値のクラスおよびサブクラス部分を定義する文書を示すストリングです。 CLI は常に SQL_DIAG_CLASS_ORIGIN フィールドに空ストリングを返します。
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	SQL_DIAG_ROW_NUMBER フィールドの値が行セットまたはパラメーター・セットで有効な行番号である場合の、結果セット内の列番号を示す値です。結果セットの列番号は常に 1 で始まります。この状況レコードがブックマーク列に関するものであれば、フィールドの値はゼロになる可能性があります。状況レコードが列番号に関連していない場合には、フィールド値は SQL_NO_COLUMN_NUMBER となります。CLI がこのレコードに関連している列番号を判別できなければ、フィールド値は SQL_COLUMN_NUMBER_UNKNOWN となります。このフィールドでは、値はステートメント・ハンドルだけに定義されます。
SQL_DIAG_CONNECTION_NAME	CHAR *	診断レコードが関連する接続の名前を示すストリングです。 CLI は常に SQL_DIAG_CONNECTION_NAME フィールドに空ストリングを返します。
SQL_DIAG_ERRMC	CHAR *	X'FF' で分けられた 1 つ以上のメッセージ・トークンを含むストリングです。
SQL_DIAG_MESSAGE_TEXT	CHAR *	エラーまたは警告に関する通知メッセージです。
SQL_DIAG_NATIVE	SQLINTEGER	ドライバーまたはデータ・ソースに固有のネイティブ・エラー・コード (存在する場合)。存在しなければ、ドライバーは 0 を返します。

表 174. DiagIdentifier 引数のレコード・フィールド (続き)

レコード・フィールド	戻りのタイプ	説明
SQL_DIAG_ROW_NUMBER	SQLINTEGER	状況レコードの関連付けに使用される、行セット内の行番号、またはパラメーター・セット内のパラメーター番号です。この状況レコードが行番号に関連していない場合には、フィールド値は SQL_NO_ROW_NUMBER となります。 CLI が、このレコードが関連付けられている行番号を判別できなければ、このフィールド値は SQL_ROW_NUMBER_UNKNOWN となります。このフィールドでは、値はステートメント・ハンドルだけに定義されます。
SQL_DIAG_SERVER_NAME	CHAR *	診断レコードが関連するサーバー名を示すストリングです。このストリングは、*InfoType* 引数に SQL_DATA_SOURCE_NAME の値を指定した SQLGetInfo() 関数の呼び出しで返される値と同じです。環境ハンドルに関連する診断データ構造の場合、およびどのサーバーにも関連しない診断の場合、このフィールドはゼロ長ストリングです。
SQL_DIAG_SQLSTATE	CHAR *	5 文字の SQLSTATE 診断コードです。
SQL_DIAG_SUBCLASS_ORIGIN	CHAR *	SQL_DIAG_CLASS_ORIGIN フィールドと同じフォーマットおよび有効値のストリングです。これは、SQLSTATE コードのサブクラスの定義部分を識別します。

動的関数フィールドの値

以下の表は、 SQL_DIAG_DYNAMIC_FUNCTION と SQL_DIAG_DYNAMIC_FUNCTION_CODE の値を示しています。これらは、SQLExecute() または SQLExecDirect() 関数への呼び出しで実行される各タイプの SQL ステートメントに適用されます。CLI はこの表の値を使用し、ODBC は他の値を指定します。

表 175. 動的関数フィールドの値

実行される SQL ステートメント	SQL_DIAG_DYNAMIC_FUNCTION の値	SQL_DIAG_DYNAMIC_ FUNCTION_CODE の値
alter-table-statement	ALTER TABLE	SQL_DIAG_ALTER_TABLE
create-index-statement	CREATE INDEX	SQL_DIAG_CREATE_INDEX
create-table-statement	CREATE TABLE	SQL_DIAG_CREATE_TABLE
create-view-statement	CREATE VIEW	SQL_DIAG_CREATE_VIEW
cursor-specification	SELECT CURSOR	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	DYNAMIC DELETE CURSOR	SQL_DIAG_DYNAMIC_DELETE_ CURSOR
delete-statement-searched	DELETE WHERE	SQL_DIAG_DELETE_WHERE
drop-index-statement	DROP INDEX	SQL_DIAG_DROP_INDEX

表 175. 動的関数フィールドの値 (続き)

実行される SQL ステートメント	SQL_DIAG_DYNAMIC_FUNCTION の値	SQL_DIAG_DYNAMIC_ FUNCTION_CODE の値
drop-table-statement	DROP TABLE	SQL_DIAG_DROP_TABLE
drop-view-statement	DROP VIEW	SQL_DIAG_DROP_VIEW
grant-statement	GRANT	SQL_DIAG_GRANT
insert-statement	INSERT	SQL_DIAG_INSERT
ODBC-procedure-extension	CALL	SQL_DIAG_PROCEDURE_CALL
revoke-statement	REVOKE	SQL_DIAG_REVOKE
update-statement-positioned	DYNAMIC UPDATE CURSOR	SQL_DIAG_DYNAMIC_UPDATE_ CURSOR
update--statement-searched	UPDATE WHERE	SQL_DIAG_UPDATE_WHERE
merge-statement	MERGE	SQL_DIAG_MERGE
不明	空ストリング	SQL_DIAG_UNKNOWN_ STATEMENT

第 7 章 CLI データ・タイプの属性

CLI アプリケーション用の SQL 記号データ・タイプおよびデフォルト・データ・タイプ

下記の表では、コール・レベル・インターフェース (CLI) アプリケーションによって使用される各 SQL データ・タイプ、それに対応する記号名、およびデフォルトの C 記号名をリストしています。

SQL データ・タイプ
この列には SQL CREATE ステートメントに表示される SQL データ・タイプを示します。SQL データ・タイプは DBMS に従属します。

記号 SQL データ・タイプ
この列には、整数値として (sqlcli.h で) 定義される SQL 記号名を示します。この値は、最初の列にリストした SQL データ・タイプを識別するために、さまざまな関数によって使用されます。

デフォルトの C 記号データ・タイプ
この列には C 記号名を示してあります。この値も整数値として定義されています。この値は、C データ・タイプを識別するために、さまざまな関数の引数で使用されます。記号名は、SQLBindParameter()、SQLGetData()、および SQLBindCol() などのさまざまな関数によってアプリケーション変数の C データ・タイプを識別する際に使用されます。これらの関数の呼び出し時に C データ・タイプを明示的に識別する代わりに、SQL_C_DEFAULT を指定することもでき、その場合 CLI は、以下の表で示したパラメーターまたは列の SQL データ・タイプに基づくデフォルト C データ・タイプを想定します。例えば、SQL_DECIMAL のデフォルト C データ・タイプは SQL_C_CHAR です。

アプリケーションでは、C データ・タイプを定義するために SQL_C_DEFAULT データ・タイプを使用しないでください。この方法は、CLI にはあまり効果的ではありません。アプリケーション内で C データ・タイプを明示的に指定すると、SQL_C_DEFAULT を使用するよりもパフォーマンスが良いので、この方法を使用することが望ましいといえます。

表 176. SQL の記号データ・タイプとデフォルト・データ・タイプ

SQL データ・タイプ	記号 SQL データ・タイプ	デフォルト記号 C データ・タイプ
BIGINT	SQL_BIGINT	SQL_C_SBIGINT
BINARY	SQL_BINARY	SQL_C_BINARY
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR [a]	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
BOOLEAN [d]	SQL_BOOLEAN	SQL_C_DEFAULT
CHAR	SQL_CHAR	SQL_C_CHAR
CHAR	SQL_TINYINT	SQL_C_TINYINT
CHAR FOR BIT DATA [b]	SQL_BINARY	SQL_C_BINARY
CHAR FOR BIT DATA	SQL_BIT	SQL_C_BINARY

表 176. SQL の記号データ・タイプとデフォルト・データ・タイプ (続き)

SQL データ・タイプ	記号 SQL データ・タイプ	デフォルト記号 C データ・タイプ
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR [a]	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
CURSOR [d]	SQL_CURSORHANDLE	SQL_C_DEFAULT
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR
DBCLOB LOCATOR [a]	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DECFLOAT(16)	SQL_DECFLOAT	SQL_C_CHAR
DECFLOAT(34)	SQL_DECFLOAT	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR [b]	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA [b]	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC [b]	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
LONG VARGRAPHIC [b]	SQL_WLONGVARCHAR	SQL_C_DBCHAR
NUMERIC [c]	SQL_NUMERIC [c]	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
ROW [d]	SQL_ROW	SQL_C_DEFAULT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
TIMESTAMP WITH TIMEZONE	SQL_TYPE_TIMESTAMP_ WITH_ TIMEZONE	SQL_C_TYPE_TIMESTAMP_ EXT_TZ
VARBINARY	SQL_VARBINARY	SQL_C_BINARY
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA [b]	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR
VARGRAPHIC	SQL_WVARCHAR	SQL_C_DBCHAR
WCHAR	SQL_WCHAR	SQL_C_WCHAR
XML[e]	SQL_XML	SQL_C_BINARY

表 176. SQL の記号データ・タイプとデフォルト・データ・タイプ (続き)

SQL データ・タイプ	記号 SQL データ・タイプ	デフォルト記号 C データ・タイプ

- **a** LOB ロケーター・タイプは持続性のある SQL データ・タイプではありません。列はロケーター・タイプでは定義できません。このタイプはパラメーター・マーカーの記述か LOB 値の表記にのみ使用されます。
- **b** LONG データ・タイプおよび FOR BIT DATA データ・タイプは、可能であれば必ず該当する LOB タイプに置き換えてください。
- **c** DB2 for z/OS (バージョン 9.1 以降)、 DB2 Server for VSE & VM、および DB2 Database for Linux, UNIX, and Windows では、NUMERIC は DECIMAL の同義語です。
- **d** BOOLEAN、CURSOR、および ROW タイプは、データベース表列またはプロシージャー・パラメーターに関する正しい DESCRIBE 情報をアプリケーションに提供する目的でのみサポートされます。これらのタイプについてバインド・インおよびバインド・アウトはサポートされません。これらのタイプを認識するのは、CLI/ODBC API の `SQLExtendedDescribe()` と `SQLDescribeParam()` のみです。
- **e** DB2 バージョン 9.7 フィックスパック 5 では、SQL_XML データ・タイプは DB2 for i V7R1 サーバーまたはそれ以降のリリースでサポートされています。
- **f** DB2 バージョン 9.7 フィックスパック 6 以降、SQL_BINARY および SQL_VARBINARY データ・タイプは、DB2 for i バージョン 6 リリース 1 サーバー以降のリリースでサポートされています。

注:

データ・タイプ DATE、DECIMAL、DECFLOAT(16)、DECFLOAT(34)、NUMERIC、TIME、および TIMESTAMP は、変換せずにデフォルト C バッファー・タイプに転送することはできません。

CLI アプリケーション用の C データ・タイプ

次の表は、CLI アプリケーションで使用されるそれぞれの記号 C タイプごとに総称型 (generic type) 定義をリストしています。

C 記号データ・タイプ
この列には C 記号名を示してあります。これは整数値として定義されています。この値は、最後の列に示された C データ・タイプを識別するために、さまざまな関数の引数で使用されます。

C タイプ
この列には C 定義済みタイプを示してあります。これは C `typedef` ステートメントを使用して `sqlcli.h` で定義されるものです。この列にある値を使用して、アプリケーションの可搬性を高めるために、すべての CLI 関連の変数および引数を宣言します。関数の引数に使用される追加の記号データ・タイプのリストについては、 583 ページの表 179 を参照してください。

基本 C タイプ
この列は参考のためにのみ示してあります。基本 C タイプはプラットフォームに依存するため、すべての変数および引数は、前の列にある記号タイプを使用して定義します。一部の値は、 581 ページの表 178 で記述されている C 構造体です。

表 177. C データ・タイプ

C 記号データ・タイプ	C タイプ	基本 C タイプ
SQL_C_BINARY	SQLCHAR	unsigned char

表 177. C データ・タイプ (続き)

C 記号データ・タイプ	C タイプ	基本 C タイプ
SQL_C_BINARYXML	SQLCHAR	unsigned char
SQL_C_BIT	SQLCHAR	unsigned char または char (値 1 または 0)
SQL_C_BLOB_LOCATOR [a]	SQLINTEGER	32 ビット整数
SQL_C_CLOB_LOCATOR [a]	SQLINTEGER	32 ビット整数
SQL_C_CHAR	SQLCHAR	unsigned char
SQL_C_DBCHAR	SQLDBCHAR	wchar_t
SQL_C_DBCLOB_LOCATOR	SQLINTEGER	32 ビット整数
SQL_C_DECIMAL64	SQLDECIMAL64	581 ページの表 178 を参照
SQL_C_DECIMAL128	SQLDECIMAL128	581 ページの表 178 を参照
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_FLOAT	SQLREAL	float
SQL_C_LONG	SQLINTEGER	32 ビット整数
SQL_C_NUMERIC [b]	SQL_NUMERIC_STRUCT	581 ページの表 178 を参照
SQL_C_SBIGINT	SQLBIGINT	64 ビット整数
SQL_C_SHORT	SQLSMALLINT	16 ビット整数
SQL_C_TINYINT	SQLSCHAR	signed char (範囲: -128 以上 127 以下)
SQL_C_TYPE_DATE	DATE_STRUCT	581 ページの表 178 を参照
SQL_C_TYPE_TIME	TIME_STRUCT	581 ページの表 178 を参照
SQL_C_TYPE_TIMESTAMP	TIMESTAMP_STRUCT	581 ページの表 178 を参照
SQL_C_TYPE_TIMESTAMP_EXT	TIMESTAMP_STRUCT_EXT	581 ページの表 178 を参照
SQL_C_TYPE_TIMESTAMP_EXT_TZ	TIMESTAMP_STRUCT_EXT_TZ	581 ページの表 178 を参照
SQL_C_UBIGINT	SQLUBIGINT	符号なし 64 ビット整数
SQL_C_ULONG	SQLUINTEGER	符号なし 32 ビット整数
SQL_C_USHORT	SQLUSMALLINT	符号なし 16 ビット整数
SQL_C_UTINYINT	SQLUCHAR	unsigned char
SQL_C_WCHAR	SQLWCHAR	wchar_t

- **a** LOB ロケーター・タイプ。
- **b** Windows のみ。

注: CLI では、SQL ファイル参照データ・タイプ (組み込み SQL で使用される) は必要ありません。
注: バイナリー XML データ送信フォーマットを利用して SQL_C_BINARYXML C データ・タイプを使用します。これは、DB2 z/OS バージョン 9.7 フィックスパック 1、および DB2 for Linux, UNIX, and Windows バージョン 9.7 フィックスパック 5 以降の DB2 CLI でサポートされます。DB2 サーバーも、バイナリー XML フォーマットをサポートするレベルである必要があります。

表 178. C 構造体

C タイプ	一般的な構造体	Windows での構造体
DATE_STRUCT	typedef struct DATE_STRUCT { SQLSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; } DATE_STRUCT;	typedef struct tagDATE_STRUCT { SWORD year; UWORD month; UWORD day; } DATE_STRUCT;
TIME_STRUCT	typedef struct TIME_STRUCT { SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; } TIME_STRUCT;	typedef struct tagTIME_STRUCT { UWORD hour; UWORD minute; UWORD second; } TIME_STRUCT;
TIMESTAMP_STRUCT	typedef struct TIMESTAMP_STRUCT { SQLUSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; SQLINTEGER fraction; } TIMESTAMP_STRUCT;	typedef struct tagTIMESTAMP_STRUCT { SWORD year; UWORD month; UWORD day; UWORD hour; UWORD minute; UWORD second; UDWORD fraction; } TIMESTAMP_STRUCT;
TIMESTAMP_STRUCT_EXT	typedef struct TIMESTAMP_STRUCT_EXT { SQLSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; SQLUINTEGER fraction; /* 1-9 digits */ SQLUINTEGER fraction2; /* 10-12 digits */ } TIMESTAMP_STRUCT_EXT;	(Windows 構造はありません。一般的な構造体だけです。)

TIMESTAMP_STRUCT_EXT_TZ	typedef struct TIMESTAMP_STRUCT
	{
	SQLSMALLINT year;
	SQLUSMALLINT month;
	SQLUSMALLINT day;
	SQLUSMALLINT hour;
	SQLUSMALLINT minute;
	SQLUSMALLINT second;
	SQLUINTEGER fraction;
	SQLUINTEGER fraction2;
	SQLSMALLINT timezone_hour;
	/*-12 to 14*/
	SQLUSMALLINT timezone_minute;
	/*-59 to 59*/
	}
	TIMESTAMP_STRUCT_EXT_TZ;
		(Windows 構造はありません。一般的な構造体だけです。)

表 178. C 構造体 (続き)

C タイプ	一般的な構造体	Windows での構造体

SQLDECIMAL64	typedef struct tagSQLDECIMAL64
	{
	union {
	SQLDOUBLE dummy;
	SQLCHAR dec64
	[SQL_DECFLOAT16_
	COEFFICIENT_LEN];
	} udec64;
	} SQLDECIMAL64;
		(Windows 構造はありません。一般的な構造体だけです。)

SQLDECIMAL128	typedef struct tagSQLDECIMAL128
	{
	union {
	SQLDOUBLE dummy;
	SQLCHAR dec128
	[SQL_DECFLOAT34_
	COEFFICIENT_LEN];
	} udec128;
	} SQLDECIMAL128;
		(Windows 構造はありません。一般的な構造体だけです。)

SQL_NUMERIC_STRUCT	(一般的な構造体はありません。 Windows 構造だけです。)	typedef struct tagSQL_NUMERIC_STRUCT { SQLCHAR precision; SQLCHAR scale; SQLCHAR sign; [a] SQLCHAR val[SQL_MAX_NUMERIC_LEN];[b] [c] } SQL_NUMERIC_STRUCT;

SQLUSMALLINT C データ・タイプに関する詳細は、 583 ページの表 179 を参照してください。

- **a** 符号付きフィールド: 1 = 正数、2 = 負数
- **b** 数値は、位取り整数として、 SQL_NUMERIC_STRUCT 構造体の値フィールドにリトル・エンディアン・モード (重要性の最も低いバイトが左端のバイトになる) で保管されます。例えば、数値 10.001 基数 10 に 4 桁の位取りを指定すると、100010 という整数に位取りされます。この数値は 16 進形式では 186AA なので、 SQL_NUMERIC_STRUCT の値は 『AA 86 01 00 00 ... 00』 となり、そのバイト数は SQL_MAX_NUMERIC_LEN #define によって定義されます。
- **c** SQL_C_NUMERIC データ・タイプの精度および位取りフィールドは、アプリケーションからの入力には使用されず、アプリケーションのドライバーからの出力のみに使用されます。ドライバーは、 SQL_NUMERIC_STRUCT に数値を書き込むときに独自のデフォルト値を精度フィールドの値として使用し、アプリケーション記述子 (デフォルト設定は 0) の SQL_DESC_SCALE フィールドの値を位取りフィールドに使用します。アプリケーションは、アプリケーション記述子の SQL_DESC_PRECISION および SQL_DESC_SCALE フィールドを設定して、精度および位取りに指定する独自の値を提供することができます。

SQL データ・タイプにマップするデータ・タイプに加えて、ポインターやハンドルなどの他の関数の引数に使用される C 記号タイプもあります。次の表に、総称データ・タイプと ODBC データ・タイプの両方を示します。

注: 製品に付属する 2 種類のドライバーがあります。それは、CLI ドライバーと 64 ビット ODBC ドライバーです。 64 ビット ODBC ドライバーは、さまざまな ODBC マネージャーにおける型定義との相違点を扱います。

表 179. C データ・タイプおよび基本 C データ・タイプ

定義済み C タイプ	基本 C タイプ	一般的な使用法
SQLPOINTER	void *	データおよびパラメーターの記憶域を指すポインター。
SQLHANDLE	1. 1. void * 2. 32 ビット整数	ハンドル情報の全 4 タイプを参照するハンドル。 1. Windows 64 ビット ODBC ドライバーと UNIX 64 ビット ODBC ドライバーの 64 ビットの値。 2. すべての 32 ビット・プラットフォームおよび 64 ビット CLI ドライバーの 32 ビットの値。
SQLHENV	1. 1. void * 2. 32 ビット整数	環境情報を参照するハンドル。 1. Windows 64 ビット ODBC ドライバーと UNIX 64 ビット ODBC ドライバーの 64 ビットの値。 2. すべての 32 ビット・プラットフォームおよび 64 ビット CLI ドライバーの 32 ビットの値。
SQLHDBC	1. 1. void * 2. 32 ビット整数	データベース接続情報を参照するハンドル。 1. Windows 64 ビット ODBC ドライバーと UNIX 64 ビット ODBC ドライバーの 64 ビットの値。 2. すべての 32 ビット・プラットフォームおよび 64 ビット CLI ドライバーの 32 ビットの値。
SQLHSTMT	1. 1. void * 2. 32 ビット整数	ステートメント情報を参照するハンドル。 1. Windows 64 ビット ODBC ドライバーと UNIX 64 ビット ODBC ドライバーの 64 ビットの値。 2. すべての 32 ビット・プラットフォームおよび 64 ビット CLI ドライバーの 32 ビットの値。
SQLUSMALLINT	符号なし 16 ビット整数	符号なしの短整数値用の関数入力引数。
SQLUINTEGER	符号なし 32 ビット整数	符号なしの長整数値用の関数入力引数。
SQLRETURN	16 ビット整数	CLI 関数からの戻りコード。
SQLULEN	1. 符号なし 64 ビット整数 2. 符号なし 32 ビット整数	1. 符号なし 64 ビット整数値の関数入力または出力引数 (64 ビット ODBC ドライバー)。 2. 符号なし 32 ビット整数値の関数入力または出力引数 (その他のすべてのドライバー)。
SQLLEN	1. 64 ビット整数 2. 32 ビット整数	1. 64 ビット整数値の関数入力または出力引数 (64 ビット ODBC ドライバー)。 2. 32 ビット整数値の関数入力または出力引数 (その他のすべてのドライバー)。

表 179. C データ・タイプおよび基本 C データ・タイプ (続き)

定義済み C タイプ	基本 C タイプ	一般的な使用法
SQLSETPOSIROW	1. 64 ビット整数 2. 16 ビット整数	1. 64 ビット整数値の関数入力または出力引数 (Windows 64 ビット ODBC ドライバー)。 2. 16 ビット整数値の関数入力または出力引数 (その他のすべてのドライバー)。

CLI でサポートされているデータ変換

以下の表には、CLI でサポートされているデータ・タイプ変換がリストされています。

最初の列には SQL データ・タイプが入ります。その後の列は C データ・タイプを示しています。 C データ・タイプの列には、次のものが入ります。

D 変換はサポートされており、SQL データ・タイプのデフォルト変換です。

X すべての IBM DBMS が変換をサポートします。

ブランク
いずれの IBM DBMS も変換をサポートしません。

データ・タイプの形式についての詳細は、データ・タイプ属性 (精度、スケール、長さ、および表示) の表を参照してください。

表 180. サポートされるデータ変換

SQL データ・タイプ	SQL_CHAR	SQL_WCHAR	SQL_BINARYXML	SQL_DBCHAR	SQL_BIT	SQL_TINYINT	SQL_STINYINT	SQL_UTINYINT	SQL_SHORT	SQL_SSHORT	SQL_USHORT	SQL_LONG	SQL_SLONG	SQL_ULONG	SQL_SBIGINT	SQL_UBIGINT	SQL_FLOAT	SQL_DOUBLE	SQL_NUMERIC	SQL_DECIMAL_IBM	SQL_DECIMAL64	SQL_DECIMAL128	SQL_DATE	SQL_TYPE_DATE	SQL_TIME	SQL_TYPE_TIME	SQL_TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_DATALINK	SQL_BINARY	SQL_CLOB_LOCATOR	SQL_BLOB_LOCATOR	SQL_DBCLOB_LOCATOR
SQL_CHAR	D	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X		X				
SQL_BINARYXML			D																													
SQL_WCHAR[4]	D	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X		X				
SQL_ VARCHAR	D	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X		X				
SQL_ WVARCHAR[4]	D	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X		X				
SQL_ LONG VARCHAR	D	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X		X				

表 180. サポートされるデータ変換 (続き)

SQL データ・タイプ	SQL_CHAR	SQL_WCHAR	SQL_BINARY_XML	SQL_DBCHAR	SQL_BIT	SQL_TINYINT	SQL_STINYINT	SQL_UTINYINT	SQL_SHORT	SQL_SSHORT	SQL_USHORT	SQL_LONG	SQL_SLONG	SQL_ULONG	SQL_SBIGINT	SQL_UBIGINT	SQL_FLOAT	SQL_DOUBLE	SQL_NUMERIC	SQL_DECIMAL_IBM	SQL_DECIMAL	SQL_DECIMAL_IBM	SQL_DATE	SQL_TYPE_TIME	SQL_TYPE_TIMESTAMP	SQL_DATALINK	SQL_BINARY	SQL_CLOB_LOCATOR	SQL_BLOB_LOCATOR	SQL_DBCLOB_LOCATOR
SQL_BINARY	X	X																									D			
SQL_VARBINARY	X	X																									D			
SQL_LONG VARBINARY	X	X																									D			
SQL_GRAPHIC	X	X		D					X		X	X		X	X	X	X	X									X			
SQL_VARGRAPHIC	X	X		D					X		X	X		X	X	X	X	X									X			
SQL_LONG VARGRAPHIC	X	X		D					X		X	X		X	X	X	X	X									X			
SQL_CLOB	D	X																									X	X		
SQL_BLOB	X	X																									D		X	
SQL_DBCLOB	X	X		D																							X			X
SQL_CLOB_LOCATOR																												D		
SQL_BLOB_LOCATOR																													D	
SQL_DBCLOB_LOCATOR																														D
SQL_NUMERIC	D	X		X[5]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					X			
SQL_DECIMAL	D	X		X[5]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					X			
SQL_DECFLOAT	D	X		X[5]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X								
SQL_INTEGER	X	X		X[5]	X	X	X	X	X	X	X	D[3]	D[3]	X	X	X	X	X	X		X	X					X			
SQL_SMALLINT	X	X		X[5]	X	X	X	X	D[2]	D[2]	X	X	X	X	X	X	X	X	X		X	X					X			

表 180. サポートされるデータ変換 (続き)

SQL データ・タイプ	SQL_C_CHAR	SQL_C_WCHAR	SQL_C_BINARYXML	SQL_C_DBCHAR	SQL_C_BIT	SQL_C_TINYINT	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_SHORT	SQL_C_SSHORT	SQL_C_USHORT	SQL_C_LONG	SQL_C_SLONG	SQL_C_ULONG	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_NUMERIC	SQL_C_DECIMAL_IBM	SQL_C_DECIMAL_64	SQL_C_DECIMAL_128	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_BINARY	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
SQL_FLOAT	X	X		X[5]	X	X	X	X	X	X	X	X	X	X	X	X	X	D	X		X	X				X			
SQL_DOUBLE	X	X		X[5]	X	X	X	X	X	X	X	X	X	X	X	X	X	D	X		X	X				X			
SQL_REAL	X	X		X[5]	X	X	X	X	X	X	X	X	X	X	X	X	D	X	X		X	X				X			
SQL_BIGINT	X	X		X[5]	X	X	X	X	X	X	X	X	X	X	D	X	X	X	X		X	X				X			
SQL_TINYINT	X	X			X	D[1]	D[1]	X	X	X	X	X	X	X	X	X	X	X	X		X	X				X			
SQL_BIT	X	X																								D			
SQL_TYPE_DATE	X	X																					D			X			
SQL_TYPE_TIME	X	X																						D		X			
SQL_TYPE_TIMESTAMP	X	X																					X	X	D				
SQL_XML	X	X	D	X																						D			

注:

1. C タイプ SQL_C_TINYINT は、CLI によってタイプ SQL_C_STINYINT として扱われます。そのため、どちらかのタイプをデフォルトと見なすことができます。
2. C タイプ SQL_C_SHORT は、CLI によってタイプ SQL_C_SSHORT として扱われます。そのため、どちらかのタイプをデフォルトと見なすことができます。
3. C タイプ SQL_C_LONG は、CLI によってタイプ SQL_C_SLONG として扱われます。そのため、どちらかのタイプをデフォルトと見なすことができます。
4. このタイプは、DB2 データベースではサポートされませんが、構成キーワード (MAPCharToWChar など) の設定により、サポートされるその他のタイプをこのタイプとしてクライアントに戻すことができます。
5. このタイプの変換は Unicode に限定されています。

SQL_C_BINARYXML C データ・タイプは、DB2 for z/OS および DB2 for Linux, UNIX, and Windows のバイナリー XML データで使用できます。 DB2 サーバー

も、バイナリー XML データ・タイプをサポートするレベルでなければなりません。 SQL_XML データ・タイプは、Informix データ・サーバーで使用することはサポートされていません。

- DB2 バージョン 9.7 フィックスパック 5 以降、SQL_XML データ・タイプは、DB2 for i バージョン 7 リリース 1 サーバー以降のリリースでサポートされています。
- DB2 バージョン 9.7 フィックスパック 6 以降、SQL_BINARY および SQL_VARBINARY データ・タイプは、DB2 for iバージョン 6 リリース 1 サーバー以降のリリースでサポートされています。

SQL_XML データ・タイプは、DB2 for i V7R1 サーバーまたはそれ以降のリリースでサポートされています。

CLI での SQL から C へのデータ変換

指定の SQL データ・タイプについて、次の内容がリストされています。

- 表の最初の列には、 SQLBindCol() および SQLGetData() の *fCType* 引数の有効な入力値をリストします。
- 2 番目の列では、テストの出力をリストします。このテストでは SQLBindCol() または SQLGetData() に指定されている *cbValueMax* 引数が頻繁に使用されます。ドライバーはこのテストを実行して、データを変換できるかどうかを判別します。
- 3 番目と 4 番目の列では、ドライバーがデータ変換を試行した後の、SQLBindCol() または SQLGetData() に指定されている *rgbValue* 引数と *pcbValue* 引数の値を (出力ごとに) リストします。
- 最後の列では、SQLFetch()、SQLExtendedFetch()、SQLGetData()、または SQLGetSubString() によって各出力用に戻される SQLSTATE をリストします。

表では、指定の SQL データ・タイプにとって有効になるよう ODBC で定義された変換をリストします。

SQLBindCol() または SQLGetData() の *fCType* 引数に、所定の SQL データ・タイプについて表にない値が含まれていると、 SQLFetch() または SQLGetData() は、SQLSTATE 07006 (データ・タイプ属性制約違反) を戻します。

fCType 引数に、表にはあっても、ドライバーでサポートされていない変換を指定する値が含まれていると、 SQLFetch() または SQLGetData() は、 SQLSTATE HYC00 (ドライバーが機能しない) を戻します。

表には示されていませんが、*pcbValue* 引数には、 SQL データ値が NULL のときの SQL_NULL_DATA が含まれます。データを取り出す場合に複数呼び出しが行われる際の *pcbValue* の使用法に関する説明は、 SQLGetData() を参照してください。

SQL データが文字 C データに変換される際に、 *pcbValue* に戻される文字カウントにはヌル終了バイトは含まれません。 *rgbValue* が NULL ポインターの場合、SQLBindCol() または SQLGetData() は、 SQLSTATE HY009 (無効な引数値) を戻します。

表では、次の用語が使われています。

データの長さ
指定の C データ・タイプに変換された後のデータの全長 (データがストリングに変換された場合のヌル終了バイトを除く)。これは、アプリケーションに戻される前にデータが切り捨てられる場合にも当てはまります。

有効桁 負符号 (必要な場合) および小数点の左の桁。

表示サイズ
文字形式でデータを表示するのに必要なバイトの合計数。

文字 SQL データから C データへの変換

文字 SQL データ・タイプは次のとおりです。

- SQL_CHAR
- SQL_VARCHAR
- SQL_LONGVARCHAR
- SQL_CLOB

表 181. 文字 SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	データ長 < cbValueMax	データ	データの長さ	00000
	データ長 >= cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARY	データ長 <= cbValueMax	データ	データの長さ	00000
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_FLOAT SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [c]	切り捨てられずに変換されたデータ [a]	データ	C データ・タイプのサイズ	00000
	変換され切り捨てられたデータ、ただし有効桁は失われていない [a]	データ	C データ・タイプのサイズ	01004
	データの変換で、有効桁が失われる [a]	影響なし	C データ・タイプのサイズ	22003
	データは数値でない [a]	影響なし	C データ・タイプのサイズ	22005
SQL_C_TYPE_DATE	データ値は有効な日付 [a]	データ	6 [b]	00000
	データ値は有効な日付ではない [a]	影響なし	6 [b]	22007
SQL_C_TYPE_TIME	データ値は有効な時刻 [a]	データ	6 [b]	00000
	データ値は有効な時刻ではない [a]	影響なし	6 [b]	22007
SQL_C_TYPE_TIMESTAMP	データ値は有効なタイム・スタンプ [a]	データ	16 [b]	00000
	データ値は有効なタイム・スタンプではない [a]	影響なし	16 [b]	22007
SQL_C_TIMESTAMP_EXT	データ値は有効なタイム・スタンプ [a]	データ	20	00000
	データ値は有効なタイム・スタンプではない [a]	影響なし	20	22007

表 181. 文字 SQL データから C データへの変換 (続き)

fCType	テスト	rgbValue	pcbValue	SQLSTATE

注:

[a] *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

[b] これは、対応する C データ・タイプのサイズです。

[c] SQL_C_NUMERIC は Windows プラットフォーム上でのみサポートされます。

SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

GRAPHIC SQL データから C データへの変換

GRAPHIC SQL データ・タイプは、以下のとおりです。

- SQL_GRAPHIC
- SQL_VARGRAPHIC
- SQL_LONGVARGRAPHIC
- SQL_DBCLOB

表 182. GRAPHIC SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	2 バイト文字数 * 2 <= cbValueMax	データ	データの長さ (オクテット)	00000
	2 バイト文字数 * 2 > cbValueMax	*cbValueMax* 未満の最大偶数バイトに切り捨てられたデータ	データの長さ (オクテット)	01004
SQL_C_DBCHAR	2 バイト文字数 * 2 < cbValueMax	データ	データの長さ (オクテット)	00000
	2 バイト文字数 * 2 >= cbValueMax	*cbValueMax* 未満の最大偶数バイトに切り捨てられたデータ	データの長さ (オクテット)	01004

注: SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

浮動小数点値への変換時に、結果値の非有効桁が失われていると、 SQLSTATE 22003 は戻されません。

数値 SQL データから C データへの変換

SQL 数値データ・タイプは、以下のとおりです。

- SQL_DECIMAL
- SQL_NUMERIC
- SQL_SMALLINT
- SQL_INTEGER
- SQL_BIGINT
- SQL_REAL

- SQL_DECFLOAT
- SQL_FLOAT
- SQL_DOUBLE

表 183. 数値 SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	表示サイズ < cbValueMax	データ	データの長さ	00000
	有効桁数 < cbValueMax	切り捨てデータ	データの長さ	01004
	有効桁数 >= cbValueMax	影響なし	データの長さ	22003
SQL_C_DBCHAR SQL_C_WCHAR	表示サイズ * 2 < cbValueMax	データ	データの長さ	00000
	有効桁数 * 2 < cbValueMax	切り捨てデータ	データの長さ	01004
	有効桁数 * 2 >= cbValueMax	影響なし	データの長さ	22003
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_DOUBLE SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [b]	切り捨てられずに変換されたデータ [a]	データ	C データ・タイプのサイズ	00000
	変換され切り捨てられたデータ、ただし有効桁は失われていない [a]	切り捨てデータ	C データ・タイプのサイズ	01004
	データの変換で、有効桁が失われる [a]	影響なし	C データ・タイプのサイズ	22003

注:

[a] *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

[b] SQL_C_NUMERIC は Windows プラットフォーム上でのみサポートされます。

SQLSTATE **00**000 は `SQLGetDiagRec()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

バイナリー SQL データから C データへの変換

バイナリー SQL データ・タイプは次のとおりです。

- SQL_BINARY
- SQL_VARBINARY
- SQL_LONGVARBINARY
- SQL_BLOB

表 184. バイナリー SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	(データ長) < cbValueMax	データ	データの長さ	N/A
	(データ長) >= cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARY	データ長 <= cbValueMax	データ	データの長さ	N/A
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004

注: DB2 バージョン 9.7 フィックスパック 6 以降、SQL_BINARY および SQL_VARBINARY データ・タイプは、DB2 for i バージョン 6 リリース 1 サーバー以降のリリースでサポートされています。

XML SQL データから C データへの変換

XML SQL データ・タイプは次のとおりです。

SQL_XML

表 185. XML SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	データ長 < cbValueMax	データ	データの長さ	00000
	データ長 >= cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARY	データ長 <= cbValueMax	データ	データの長さ	00000
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_BINARYXML	データ長 <= cbValueMax	データ	データの長さ	00000
	データ長 > cbValueMax	切り捨てデータ	データの長さ	01004
SQL_C_DBCHAR	2 バイト文字数 * 2 < cbValueMax	データ	データの長さ	00000
	2 バイト文字数 * 2 >= cbValueMax	cbValueMax 未満の最大偶数バイトに切り捨てられたデータ	データの長さ	01004
SQL_C_WCHAR	2 バイト文字数 * 2 < cbValueMax	データ	データの長さ	00000
	2 バイト文字数 * 2 >= cbValueMax	cbValueMax 未満の最大偶数バイトに切り捨てられたデータ	データの長さ	01004

注:

1. SQLSTATE **00**000 は `SQLGetDiagRec()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。
2. データの長さには、ターゲット・エンコード内の XML 宣言が含まれます。
3. SQL_XML データ・タイプは、Informix データ・サーバーで使用することはサポートされていません。

日付 SQL データから C データへの変換

日付 SQL データ・タイプは次のとおりです。

- SQL_TYPE_DATE

表 186. 日付 SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 11	データ	10	00000
	cbValueMax < 11	影響なし	10	22003
SQL_C_TYPE_DATE	なし [a]	データ	6 [b]	00000

表 186. 日付 SQL データから C データへの変換 (続き)

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_TYPE_TIMESTAMP	なし [a]	データ [c]	16 [b]	00000
SQL_C_TIMESTAMP_EXT	なし [a]	データ [c]	20	00000

注:

[a] *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

[b] これは、対応する C データ・タイプのサイズです。

[c] TIMESTAMP_STRUCT または TIMESTAMP_STRUCT_EXT 構造の時刻フィールドはゼロに設定されます。

SQLSTATE **00**000 は `SQLGetDiagRec()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

日付 SQL データ・タイプが文字の C データ・タイプに変換される場合、ストリングは「yyyy-mm-dd」形式になります。

時刻 SQL データから C データへの変換

時刻 SQL データ・タイプは次のとおりです。

- SQL_TYPE_TIME

表 187. 時刻 SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 9	データ	8	00000
	cbValueMax < 9	影響なし	8	22003
SQL_C_TYPE_TIME	なし [a]	データ	6 [b]	00000
SQL_C_TYPE_TIMESTAMP	なし [a]	データ [c]	16 [b]	00000
SQL_C_TIMESTAMP_EXT	なし [a]	データ [c]	20	00000

注:

[a] *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

[b] これは、対応する C データ・タイプのサイズです。

[c] TIMESTAMP_STRUCT または TIMESTAMP_STRUCT_EXT 構造の日付フィールドは、アプリケーションが実行しているマシンの現行システム日付に設定され、時刻小数部はゼロに設定されます。

SQLSTATE **00**000 は `SQLGetDiagRec()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

時刻 SQL データ・タイプが文字の C データ・タイプに変換される場合、ストリングは「hh:mm:ss」形式になります。

タイム・スタンプ SQL データから C データへの変換

タイム・スタンプ SQL データ・タイプは次のとおりです。

- SQL_TYPE_TIMESTAMP

表 188. タイム・スタンプ SQL データから C データへの変換

fCType	テスト	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	表示サイズ < cbValueMax	データ	データの長さ	00000
	19 <= cbValueMax <= 表示サイズ	切り捨てデータ [b]	データの長さ	01004
	cbValueMax < 19	影響なし	データの長さ	22003
SQL_C_TYPE_DATE	なし [a]	切り捨てデータ [c]	6 [e]	01004
SQL_C_TYPE_TIME	なし [a]	切り捨てデータ [d]	6 [e]	01004
SQL_C_TYPE_TIMESTAMP	なし [a]	データ	16 [e]	00000
SQL_C_TIMESTAMP_EXT	なし [a]	データ	20	00000

注:

[a] *cbValueMax* の値はこの変換では無視されます。ドライバーは、 *rgbValue* のサイズは C データ・タイプのサイズであると想定します。

[b] タイム・スタンプの小数秒は切り捨てられます。

[c] タイム・スタンプの時刻部分は削除されます。

[d] タイム・スタンプの日付部分は削除されます。

[e] これは、対応する C データ・タイプのサイズです。

SQLSTATE **00**000 は `SQLGetDiagRec()` では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

タイム・スタンプ SQL データ・タイプが文字の C データ・タイプに変換されるとき、結果のストリングは yyyy-mm-dd hh:mm:ss.fffffffffff の形式になります。ここで、秒の小数部分の桁数は、タイム・スタンプ SQL データ・タイプの精度には関係なく、0 から 12 です。アプリケーションに ISO フォーマットが必要な場合、CLI/ODBC 構成キーワード PATCH2=33 に設定します。

タイム・ゾーンが指定された timestamp(p) SQL データから C データへの変換

タイム・スタンプ SQL データ・タイプは次のとおりです。

- SQL_TYPE_TIMESTAMP_WITH_TIMEZONE

表 189. タイム・ゾーンが指定された timestamp(p) SQL データから C データへの変換

fCType	SQL タイプ	テスト/結果	SQLSTATE
SQL_C_TYPE_TIMESTAMP_EXT_TZ	SQL_CHAR	データ値は、タイム・ゾーンが指定された有効なタイム・スタンプ	N/A
		データ値は、タイム・ゾーンが指定された有効なタイム・スタンプではない	22007
SQL_C_TYPE_DATE	SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	切り捨てデータ	01S07

表 189. タイム・ゾーンが指定された timestamp(p) SQL データから C データへの変換 (続き)

fCType	SQL タイプ	テスト/結果	SQLSTATE
SQL_C_TYPE_TIME	SQL_TYPE_TIMESTAMP_ WITH_TIMEZONE	切り捨てデータ	01S07
SQL_C_TYPE_TIMESTAMP	SQL_TYPE_TIMESTAMP_ WITH_TIMEZONE	切り捨てデータ	01S07

SQL から C へのデータ変換例

表 190. SQL から C へのデータ変換例

SQL データ・タイプ	SQL データ値	C データ・タイプ	cbValue max	rgbValue	SQL STATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef¥0 [a]	00000
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde¥0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56¥0 [a]	00000
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234¥0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	---	22003
SQL_DECIMAL	1234.56	SQL_C_FLOAT	無視されます	1234.56	00000
SQL_DECIMAL	1234.56	SQL_C_SHORT	無視されます	1234	01004
SQL_TYPE_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31¥0 [a]	00000
SQL_TYPE_DATE	1992-12-31	SQL_C_CHAR	10	---	22003
SQL_TYPE_DATE	1992-12-31	SQL_C_TYPE_ TIMESTAMP	無視されます	1992,12,31, 0,0,0,0 [b]	00000
SQL_TYPE_ TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12¥0 [a]	00000
SQL_TYPE_ TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1¥0 [a]	01004
SQL_TYPE_ TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	---	22003

注:

[a] 「¥0」はヌル終了文字を表します。

[b] このリストの数値は、TIMESTAMP_STRUCT または TIMESTAMP_STRUCT_EXT 構造体のフィールドに保管される数値です。

SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

CLI での C から SQL へのデータ変換

指定の C データ・タイプについて、次の内容がリストされています。

- 表の最初の列には、SQLBindParameter() または SQLSetParam() の *fSqlType* 引数の有効な入力値をリストします。
- 2 番目の列では、テストの出力をリストします。このテストでは SQLBindParameter() または SQLSetParam の *pcbValue* 引数で指定されたパラメーター・データの長さが頻繁に使用されます。ドライバーはこのテストを実行して、データを変換できるかどうかを判別します。

- 3 番目の列では、 SQLExecDirect() または SQLExecute() によって各出力に返される SQLSTATE をリストします。

表では、指定の SQL データ・タイプにとって有効になるよう ODBC で定義された変換をリストします。

SQLBindParameter() または SQLSetParam() の *fSqlType* 引数に、指定の C データ・タイプについて表にない値が含まれている場合、 SQLSTATE 07006 (データ・タイプ属性制約違反) が戻されます。

fSqlType 引数に、表にはあっても、ドライバーでサポートされていない変換を指定する値が含まれていると、 SQLBindParameter() または SQLSetParam() は、SQLSTATE HYC00 (ドライバーが機能しない) を戻します。

SQLBindParameter() または SQLSetParam() に指定された *rgbValue* および *pcbValue* 引数が両方とも NULL ポインターである場合、その関数は SQLSTATE HY009 (引数値が無効です) を戻します。

注:

- SQL_XML データ・タイプは、Informix データ・サーバーで使用することはサポートされていません。
- DB2 バージョン 9.7 フィックスパック 6 以降、SQL_BINARY および SQL_VARBINARY データ・タイプは、DB2 for i バージョン 6 リリース 1 サーバー以降のリリースでサポートされています。

データの長さ
: 指定された SQL データ・タイプに変換された後のデータの全長 (データがストリングに変換された場合のヌル終了バイトを除く)。これは、データ・ソースに送られる前にデータが切り捨てられる場合にも当てはまります。

列の長さ
: データがデフォルト C データ・タイプへ転送されるときにアプリケーションに戻されるバイトの最大数。文字データの場合、長さにはヌル終了バイトは含まれません。

表示サイズ
: データを文字書式で表示するために必要な最大バイト数。

有効桁 負符号 (必要な場合) および小数点の左の桁。

文字 C データから SQL データへの変換

文字 C データ・タイプは次のとおりです。

- SQL_C_CHAR

表 191. 文字 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	データ長 <= 列長	N/A
SQL_CHAR	データ長 > 列長	22001
SQL_VARCHAR		
SQL_LONGVARCHAR		
SQL_CLOB		

表 191. 文字 C データから SQL データへの変換 (続き)

fSQLType	Test	SQLSTATE
	切り捨てられずに変換されたデータ	N/A
SQL_DECIMAL	変換され切り捨てられたデータ、ただし有効桁は失われていない	22001
SQL_NUMERIC	データ変換の結果、有効桁が消失する	22003
SQL_SMALLINT	データ値は数値ではない	22005
SQL_INTEGER		
SQL_BIGINT		
SQL_REAL		
SQL_FLOAT		
SQL_DOUBLE		
	(データ長) < 列長	N/A
SQL_BINARY	(データ長) >= 列長	22001
SQL_VARBINARY	データ値は 16 進値ではない	22005
SQL_LONGVARBINARY		
SQL_BLOB		
SQL_TYPE_DATE	データ値は有効な日付	N/A
	データ値は有効な日付ではない	22007
	データ値は有効なタイム・スタンプ	22008
	データ値は有効なタイム・スタンプで、接続属性 SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN は 1 に設定される	01S07 (小数部分の切り捨ての警告)
SQL_TYPE_TIME	データ値は有効な時刻	N/A
	データ値は有効な時刻ではない	22007
	データ値は有効なタイム・スタンプ	22008
	データ値は有効なタイム・スタンプで、接続属性 SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN は 1 に設定される	01S07 (小数部分の切り捨ての警告)
SQL_TYPE_TIMESTAMP	データ値は有効なタイム・スタンプ	N/A
	データ値は有効なタイム・スタンプではない	22007
	データ値は有効な日付	N/A
	データ長 / 2 <= 列長	N/A
SQL_GRAPHIC	データ長 / 2 < 列長	22001
SQL_VARGRAPHIC		
SQL_LONGVARGRAPHIC		
SQL_DBCLOB		
SQL_XML	データは、暗黙的に構文解析されることがあります。	(いくつかの SQLSTATES が戻されることがあります)

数値 C データから SQL データへの変換

数値 C データ・タイプは次のとおりです。

- SQL_C_SHORT
- SQL_C_LONG
- SQL_C_FLOAT
- SQL_C_DOUBLE
- SQL_C_TINYINT
- SQL_C_SBIGINT
- SQL_C_BIT

表 192. 数値 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	切り捨てられずに変換されたデータ	N/A
SQL_DECIMAL	変換され切り捨てられたデータ、ただし有効桁は失われていない	22001
SQL_NUMERIC	データ変換の結果、有効桁が消失する	22003
SQL_SMALLINT		
SQL_INTEGER		
SQL_BIGINT		
SQL_REAL		
SQL_FLOAT		
SQL_DOUBLE		
	切り捨てられずに変換されたデータ	N/A
SQL_CHAR	データ変換の結果、有効桁が消失する	22003
SQL_VARCHAR		

注: 浮動小数点値への変換時に、結果値の非有効桁が失われていると、 SQLSTATE 22003 は戻されません。

バイナリー C データから SQL データへの変換

バイナリー C データ・タイプは次のとおりです。

- SQL_C_BINARY

表 193. バイナリー C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	データ長 <= 列長	N/A
SQL_CHAR	データ長 > 列長	22001
SQL_VARCHAR		
SQL_LONGVARCHAR		
SQL_CLOB		
	データ長 <= 列長	N/A
SQL_BINARY	データ長 > 列長	22001
SQL_VARBINARY		
SQL_LONGVARBINARY		
SQL_BLOB		
SQL_XML	データは、暗黙的に構文解析されることがあります。	(いくつかの SQLSTATES が戻されることがあります)

DBCHAR C データから SQL データへの変換

2 バイト C データ・タイプは以下のとおりです。

- SQL_C_DBCHAR

表 194. DBCHAR C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	データ長 <= 列長 x 2	N/A
SQL_CHAR	データ長 > 列長 × 2	22001
SQL_VARCHAR		
SQL_LONGVARCHAR		
SQL_CLOB		
SQL_DECIMAL	データ長 <= 列長 x 2	N/A
SQL_NUMERIC	データ長 > 列長 × 2	22001
SQL_SMALLINT	データは非 Unicode	07006
SQL_INTEGER		
SQL_BIGINT		
SQL_REAL		
SQL_FLOAT		
SQL_DECFLOAT		
SQL_DOUBLE		
	データ長 <= 列長 x 2	N/A
SQL_BINARY	データ長 > 列長 × 2	22001
SQL_VARBINARY		
SQL_LONGVARBINARY		
SQL_BLOB		
SQL_XML	データは、暗黙的に構文解析されることがあります。	(いくつかの SQLSTATES が戻されることがあります)

日付 C データから SQL データへの変換

日付 C データ・タイプは次のとおりです。

- SQL_C_DATE

表 195. 日付 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	列長 >= 10	N/A
SQL_CHAR	列長 < 10	22001
SQL_VARCHAR		
SQL_TYPE_DATE	データ値は有効な日付	N/A
	データ値は有効な日付ではない	22007
SQL_TYPE_TIMESTAMP[a]	データ値は有効な日付	N/A
	データ値は有効な日付ではない	22007

表 195. 日付 C データから SQL データへの変換 (続き)

fSQLType	Test	SQLSTATE

注: SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

注: **a**、TIMESTAMP の時刻コンポーネントはゼロに設定されます。

時刻 C データから SQL データへの変換

時刻 C データ・タイプは次のとおりです。

- SQL_C_TIME

表 196. 時刻 C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	列長 >= 8	N/A
SQL_CHAR SQL_VARCHAR	列長 < 8	22001
SQL_TYPE_TIME	データ値は有効な時刻	N/A
	データ値は有効な時刻ではない	22007
SQL_TYPE_TIMESTAMP [a]	データ値は有効な時刻	N/A
	データ値は有効な時刻ではない	22007

注: SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

注: **a**、TIMESTAMP の日付コンポーネントは、アプリケーションが実行しているマシンのシステム日付に設定されます。

タイム・スタンプ C データから SQL データへの変換

タイム・スタンプ C データ・タイプは次のとおりです。

- SQL_C_TIMESTAMP

表 197. タイム・スタンプ C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	列長 >= 表示サイズ	N/A
	26 <= 列長 < 表示サイズ [a]	N/A
	列長 < 26	22001
SQL_TYPE_DATE	時刻フィールドがゼロ	N/A
	時刻フィールドがゼロ以外	22008
	データ値に有効な日付が含まれていない [b]	22007
	時刻フィールドはゼロ以外で、接続属性 SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN は 1 に設定される	01S07 (小数部分の切り捨ての警告)
SQL_TYPE_TIME	小数秒フィールドがゼロ	N/A
	小数秒フィールドがゼロ以外	22008
	データ値に有効な時刻が含まれていない	22007
	時刻フィールドはゼロ以外で、接続属性 SQL_ATTR_REPORT_TIMESTAMP_TRUNC_AS_WARN は 1 に設定される	01S07 (小数部分の切り捨ての警告)
SQL_TYPE_TIMESTAMP	データ値は有効なタイム・スタンプ	N/A
	データ値は有効なタイム・スタンプではない	22007

注:

a タイム・スタンプの小数秒は切り捨てられます。

b timestamp_struct は、時間、分、秒および小数部を 0 にリセットする必要があります。そうでない場合は、SQLSTATE 22008 が返されます。

SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

可変タイム・スタンプ C データから SQL データへの変換

タイム・スタンプ C データ・タイプは次のとおりです。

- SQL_C_TIMESTAMP_EXT

表 198. 可変タイム・スタンプ C データから SQL データへの変換

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	列長 >= 表示サイズ	N/A
	26 <= 列長 < 表示サイズ [a]	N/A
	列長 < 26	22001
	小数秒フィールド > 12	22007
SQL_TYPE_DATE	時刻フィールドがゼロ	N/A
	時刻フィールドがゼロ以外	22008
	データ値に有効な日付が含まれていない [b]	22007
SQL_TYPE_TIME	小数秒フィールドがゼロ	N/A
	小数秒フィールドがゼロ以外	22008
	データ値に有効な時刻が含まれていない	22007
SQL_TYPE_TIMESTAMP	データ値は有効なタイム・スタンプ	N/A
	データ値は有効なタイム・スタンプではない	22007
	TIMESTAMP(p) <= fractional seconds fields <= 12 [a] によって指定される精度	N/A

注:

a タイム・スタンプの小数秒は切り捨てられます。

b timestamp_struct は、時間、分、秒および小数部を 0 にリセットする必要があります。そうでない場合は、SQLSTATE 22008 が返されます。

SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

タイム・ゾーンが指定された timestamp(p) C データから SQL データへの変換

タイム・ゾーンが指定されたタイム・スタンプ C データ・タイプは次のとおりです。

- SQL_C_TIMESTAMP_EXT_TZ

表 199. タイム・ゾーンが指定されたタイム・スタンプ C データから SQL データへの変換

fSQLType	Test	SQLSTATE
	列長 >= 表示サイズ	N/A
SQL_CHAR /SQL_VARCHAR SQL_WCHAR/SQL_WVARCHAR SQL_LONGVARCHAR	列長 < 27+p	22001
SQL_TYPE_DATE	時刻フィールドがゼロ	N/A
	時刻フィールドがゼロ以外	22008
	データ値に有効な日付が含まれていない	22007
	タイム・ゾーン・フィールドがゼロ以外	22008
SQL_TYPE_TIME	小数秒フィールドがゼロ	N/A
	小数秒フィールドがゼロ以外	22008
	データ値に有効な時刻が含まれていない	22007
	タイム・ゾーン・フィールドがゼロ以外	22008
SQL_TYPE_TIMESTAMP	データ値は有効なタイム・スタンプ	N/A
	データ値は有効なタイム・スタンプではない	22007
	タイム・ゾーン・フィールドがゼロ以外	22008
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	データ値は有効なタイム・スタンプ	N/A
	データ値は、タイム・ゾーンが指定された有効なタイム・スタンプ	22007

C から SQL へのデータ変換例

表 200. C から SQL へのデータ変換例

C データ・タイプ	C データ値	SQL データ・タイプ	列の長さ	SQL データ値	SQL STATE
SQL_C_CHAR	abcdef¥0	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef¥0	SQL_CHAR	5	abcde	22001
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	6	1234.56	N/A
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	5	1234.5	22001
SQL_C_CHAR	1234.56¥0	SQL_DECIMAL	3	---	22003
SQL_C_CHAR	4.46.32	SQL_TYPE_TIME	6	4.46.32	N/A
SQL_C_CHAR	4-46-32	SQL_TYPE_TIME	6	該当せず	22007
SQL_C_DOUBLE	123.45	SQL_CHAR	22	1.23450000 000000e+02	N/A
SQL_C_FLOAT	1234.56	SQL_FLOAT	該当せず	1234.56	N/A

表 200. C から SQL へのデータ変換例 (続き)

C データ・タイプ	C データ値	SQL データ・タイプ	列の長さ	SQL データ値	SQL STATE
SQL_C_FLOAT	1234.56	SQL_INTEGER	該当せず	1234	22001
SQL_C_	1992-12-31 23:45:55. 123456	SQL_TYPE_DATE	6	1992-12-31	01004
SQL_C_ TIMESTAMP_EXT	2009-06-06 23:45:55. 123456789876	SQL_TYPE_DATE	6	2009-06-06	01004

注: SQLSTATE **00**000 は SQLGetDiagRec() では戻されません。これは、関数が SQL_SUCCESS を戻すときに示されます。

データ・タイプ属性

データ・タイプ精度 (CLI) 表

数値列またはパラメーターの精度は、その列またはパラメーターのデータ・タイプで使用される桁数の最大数を参照します。非数字の列またはパラメーターの精度とは一般的に、列またはパラメーターの文字の最大数または定義数を指します。次の表は、各 SQL データ・タイプの精度を定義しています。

表 201. 精度

fSqlType	精度
SQL_CHAR SQL_VARCHAR SQL_CLOB	列またはパラメーターの定義された長さ。例えば、CHAR(10) と定義された列の精度は 10 です。
SQL_LONGVARCHAR	列またはパラメーターの最大長。 [a]
SQL_DECIMAL SQL_DECFLOAT SQL_NUMERIC	桁数の定義された最大数。例えば、NUMERIC(10,3) と定義された列の精度は 10、DECFLOAT(34) と定義された列の精度は 34 です。
SQL_SMALLINT [b]	5
SQL_BIGINT	19
SQL_INTEGER [b]	10
SQL_FLOAT [b]	15
SQL_REAL [b]	7
SQL_DOUBLE [b]	15
SQL_BINARY SQL_VARBINARY SQL_BLOB	列またはパラメーターの定義された長さ。例えば、CHAR(10) FOR BIT DATA と定義された列の精度は 10 です。
SQL_LONGVARBINARY	列またはパラメーターの最大長。
SQL_TYPE_DATE [b]	10 (yyyy-mm-dd フォーマットの文字数)
SQL_TYPE_TIME [b]	8 (hh:mm:ss フォーマットの文字数)

表 201. 精度 (続き)

fSqlType	精度
SQL_TYPE_TIMESTAMP	TIMESTAMP データ・タイプで使用する、「yyyy-mm-dd hh:mm:ss[.fffffffffff]」フォーマットの文字数。例えば、タイム・スタンプが秒または小数秒を使用しない場合、精度は 16 です (「yyyy-mm-dd hh:mm」フォーマットの文字数)。タイム・スタンプが千分の一秒を使用する場合、精度は 23 です (「yyyy-mm-dd hh:mm:ss.fff」フォーマットの文字数)。
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	TIMESTAMP_WITH_TIMEZONE データ・タイプで使用する、「yyyy-mm-dd hh:mm:ss[.fffffffffff]」フォーマットの文字数。タイム・ゾーンの有効範囲は -12:59 から +14:00 です。timezone_hour が負の場合、timezone_minute も負かゼロにする必要があります。timezone_hour が正の場合、timezone_minute も正かゼロにする必要があります。timezone_hour がゼロの場合には、timezone_minute に指定できるのは -59 から +59 までの範囲の値です。
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	列またはパラメーターの定義された長さ。例えば、GRAPHIC(10) と定義された列の精度は 10 です。
SQL_LONGVARGRAPHIC	列またはパラメーターの最大長。
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	列またはパラメーターの定義された長さ。例えば、WCHAR(10) と定義された列の精度は 10 です。
SQL_XML	XML 値が外部ルーチンに対する引数でなければ、0。外部ルーチンの場合、精度は XML AS CLOB(n) 引数の定義済みの長さ、n です。
注: [a] このデータ・タイプのパラメーターの精度を SQLBindParameter() または SQLSetParam() で定義する場合、 *cbParamDef* は、この表で定義されている精度ではなく、データの全長に設定してください。 [b] このデータ・タイプでは、 SQLBindParameter() の *cbColDef* 引数は無視されます。	

データ・タイプ・スケール (CLI) 表

数値の列またはパラメーターのスケールは、小数点の右にある桁の最大数を参照します。近似の浮動小数点数の列またはパラメーターの場合、小数点の右の桁数が固定されないため、スケールは定義されないことに注意してください。次の表は、各 SQL データ・タイプのスケールを定義しています。

表 202. スケール

fSqlType	スケール
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	該当しません。
SQL_DECIMAL SQL_NUMERIC	小数点の右の定義された桁数。例えば、NUMERIC (10, 3) と定義された列のスケールは 3 です。
SQL_SMALLINT SQL_INTEGER SQL_BIGINT	0
SQL_REAL SQL_FLOAT SQL_DECFLOAT SQL_DOUBLE	該当しません。
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	該当しません。
SQL_TYPE_DATE SQL_TYPE_TIME	該当しません。
SQL_TYPE_TIMESTAMP	「yyyy-mm-dd hh:mm:ss[.fffffffffff]」フォーマットの、小数点の右にある桁の数。例えば、TIMESTAMP データ・タイプが「yyyy-mm-dd hh:mm:ss.fff」フォーマットを使用する場合、スケールは 3 です。
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLOB	該当しません。
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	該当しません。
SQL_XML	該当しません。

データ・タイプ長 (CLI) 表

列の長さとは、データがデフォルト C データ・タイプに転送されるときにアプリケーションに戻される最大バイト 数のことです。文字データの場合、長さにはヌル終了バイトは含まれません。列の長さは、データ・ソースにデータを保管するのに必要なバイト数とは違う場合があることに注意してください。

次の表は、各 SQL データ・タイプの長さを定義しています。

表 203. 長さ

fSqlType	長さ
SQL_CHAR SQL_VARCHAR SQL_CLOB	列の定義された長さ。例えば、CHAR(10) と定義された列の長さは 10 です。
SQL_LONGVARCHAR	列の最大長。
SQL_DECIMAL SQL_NUMERIC	最大桁数に 2 を加えた値。このデータ・タイプは文字ストリングとして戻されるので、桁数、符号、および小数点の文字が必要です。例えば、NUMERIC(10,3) と定義された列の長さは 12 です。
SQL_DECFLOAT	列が DECFLOAT(16) として定義されている場合、長さは 8 です。列が DECFLOAT(34) として定義されている場合、長さは 16 です。
SQL_SMALLINT	2 (2 バイト)。
SQL_INTEGER	4 (4 バイト)。
SQL_BIGINT	8 (8 バイト)。
SQL_REAL	4 (4 バイト)。
SQL_FLOAT	8 (8 バイト)。
SQL_DOUBLE	8 (8 バイト)。
SQL_BINARY SQL_VARBINARY SQL_BLOB	列の定義された長さ。例えば、CHAR(10) FOR BIT DATA と定義された列の長さは 10 です。
SQL_LONGVARBINARY	列の最大長。
SQL_TYPE_DATE SQL_TYPE_TIME	6 (DATE_STRUCT または TIME_STRUCT 構造のサイズ)。
SQL_TYPE_TIMESTAMP	16 (TIMESTAMP_STRUCT 構造のサイズ)。
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	列の定義された長さの 2 倍。例えば、GRAPHIC(10) と定義された列の長さは 20 です。
SQL_LONGVARGRAPHIC	列の最大長の 2 倍。
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	列の定義された長さの 2 倍。例えば、WCHAR(10) と定義された列の長さは 20 です。

表 203. 長さ (続き)

fSqlType	長さ
SQL_XML	0 (ただし、保管される XML 文書のサイズは 2GB に制限されています)。

データ・タイプ表示 (CLI) 表

列の表示サイズとは、文字形式でデータを表示するのに必要な最大バイト 数のことです。次の表は、各 SQL データ・タイプの表示サイズを定義しています。

表 204. 表示サイズ

fSqlType	表示サイズ
SQL_CHAR SQL_VARCHAR SQL_CLOB	列の定義された長さ。例えば、CHAR(10) と定義された列の表示サイズは 10 です。
SQL_LONGVARCHAR	列の最大長。
SQL_DECIMAL SQL_NUMERIC	列の精度に 2 を加えたもの (符号、精度の桁数、および小数点)。例えば、NUMERIC(10,3) と定義された列の表示サイズは 12 です。
SQL_DECFLOAT	列が DECFLOAT(16) として定義されている場合、表示の長さは 24 です。列が DECFLOAT(34) として定義されている場合、表示の長さは 42 です。
SQL_SMALLINT	6 (符号および 5 桁)。
SQL_INTEGER	11 (符号および 10 桁)。
SQL_BIGINT	20 (符号および 19 桁)。
SQL_REAL	13 (符号、7 桁、小数点、文字 E、符号、および 2 桁)。
SQL_FLOAT SQL_DOUBLE	22 (符号、15 桁、小数点、文字 E、符号、および 3 桁)。
SQL_BINARY SQL_VARBINARY SQL_BLOB	列の定義された最大長の 2 倍 (それぞれのバイナリー・バイトは 2 桁の 16 進数で表されます)。例えば、CHAR(10) FOR BIT DATA と定義された列の表示サイズは 20 です。
SQL_LONGVARBINARY	列の最大長の 2 倍。
SQL_TYPE_DATE	10 (yyyy-mm-dd フォーマットの日付)。
SQL_TYPE_TIME	8 (hh:mm:ss フォーマットの時刻)。

表 204. 表示サイズ (続き)

fSqlType	表示サイズ
SQL_TYPE_TIMESTAMP	19 (タイム・スタンプのスケールが 0 の場合)、または 20 にタイム・スタンプのスケールを加えたもの (スケールが 0 より大きい場合)。これは、「yyyy-mm-dd hh:mm:ss[.fffffffffff]」フォーマットの文字数です。例えば、千分の一秒を保管する列の表示サイズは 23 です (「yyyy-mm-dd hh:mm:ss.fff」の文字数)。
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	列またはパラメーターの定義された長さの 2 倍。例えば、GRAPHIC(10) と定義された列の表示サイズは 20 です。
SQL_LONGVARGRAPHIC	列またはパラメーターの最大長。
SQL_XML	0

付録 A. DB2 技術情報の概説

DB2 技術情報は、さまざまな方法でアクセスすることが可能な、各種形式で入手できます。

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、IBM Publications Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss) から利用できる DB2 ライブラリーについて説明しています。英語および翻訳された DB2 バージョン 10.1 のマニュアル (PDF 形式) は、www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: *DB2 インフォメーション・センター*は、PDF やハードコピー資料よりも頻繁に更新されます。

表 205. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
管理 API リファレンス	SA88-4671-00	入手可能	2012 年 4 月
管理ルーチンおよびビュー	SA88-4672-00	入手不可	2012 年 4 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SA88-4676-00	入手可能	2012 年 4 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SA88-4677-00	入手可能	2012 年 4 月
コマンド・リファレンス	SA88-4673-00	入手可能	2012 年 4 月
データベース: 管理の概念および構成リファレンス	SA88-4662-00	入手可能	2012 年 4 月
データ移動ユーティリティー ガイドおよびリファレンス	SA88-4693-00	入手可能	2012 年 4 月
データベースのモニタリング ガイドおよびリファレンス	SA88-4663-00	入手可能	2012 年 4 月
データ・リカバリーと高可用性 ガイドおよびリファレンス	SA88-4694-00	入手可能	2012 年 4 月
データベース・セキュリティー・ガイド	SA88-4695-00	入手可能	2012 年 4 月

表 205. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 ワークロード管理ガイドおよびリファレンス	SA88-4685-00	入手可能	2012 年 4 月
ADO.NET および OLE DB アプリケーションの開発	SA88-4665-00	入手可能	2012 年 4 月
組み込み SQL アプリケーションの開発	SA88-4666-00	入手可能	2012 年 4 月
Java アプリケーションの開発	SA88-4669-00	入手可能	2012 年 4 月
Perl、PHP、Python および Ruby on Rails アプリケーションの開発	SA88-4670-00	入手不可	2012 年 4 月
SQL および外部ルーチンの開発	SA88-4667-00	入手可能	2012 年 4 月
データベース・アプリケーション開発の基礎	GI88-4279-00	入手可能	2012 年 4 月
DB2 インストールおよび管理 概説 (Linux および Windows 版)	GI88-4280-00	入手可能	2012 年 4 月
グローバリゼーション・ガイド	SA88-4696-00	入手可能	2012 年 4 月
DB2 サーバー機能 インストール	GA88-4679-00	入手可能	2012 年 4 月
IBM データ・サーバー・クライアント機能 インストール	GA88-4680-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 1 巻	SA88-4688-00	入手不可	2012 年 4 月
メッセージ・リファレンス 第 2 巻	SA88-4689-00	入手不可	2012 年 4 月
Net Search Extender 管理およびユーザーズ・ガイド	SA88-4691-00	入手不可	2012 年 4 月
パーティションおよびクラスタリングのガイド	SA88-4697-00	入手可能	2012 年 4 月
pureXML ガイド	SA88-4686-00	入手可能	2012 年 4 月
Spatial Extender ユーザーズ・ガイドおよびリファレンス	SA88-4690-00	入手不可	2012 年 4 月

表 205. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート	SA88-4668-00	入手可能	2012 年 4 月
SQL リファレンス 第 1 巻	SA88-4674-00	入手可能	2012 年 4 月
SQL リファレンス 第 2 巻	SA88-4675-00	入手可能	2012 年 4 月
Text Search ガイド	SA88-4692-00	入手可能	2012 年 4 月
問題判別およびデータベース・パフォーマンスのチューニング	SA88-4664-00	入手可能	2012 年 4 月
DB2 バージョン 10.1 へのアップグレード	SA88-4678-00	入手可能	2012 年 4 月
DB2 バージョン 10.1 の新機能	SA88-4684-00	入手可能	2012 年 4 月
XQuery リファレンス	SA88-4687-00	入手不可	2012 年 4 月

表 206. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Connect DB2 Connect Personal Edition インストールおよび構成	SA88-4681-00	入手可能	2012 年 4 月
DB2 Connect DB2 Connect サーバー機能インストールおよび構成	SA88-4682-00	入手可能	2012 年 4 月
DB2 Connect ユーザーズ・ガイド	SA88-4683-00	入手可能	2012 年 4 月

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 製品は、SQL ステートメントの結果の原因になったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順

SQL 状態ヘルプを開始するには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate または ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。
例えば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

他のバージョンの DB2 製品の資料は、ibm.com® のそれぞれのインフォメーション・センターにあります。

このタスクについて

DB2 バージョン 10.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v10r1 です。

DB2 バージョン 9.8 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/ です。

DB2 バージョン 9.7 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/ です。

DB2 バージョン 9.5 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5 です。

DB2 バージョン 9.1 のトピックを扱っている *DB2 インフォメーション・センター* の URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9/ です。

DB2 バージョン 8 のトピックについては、*DB2 インフォメーション・センター* の URL (http://publib.boulder.ibm.com/infocenter/db2luw/v8/) を参照してください。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

ローカルにインストールした DB2 インフォメーション・センターは、定期的に更新する必要があります。

始める前に

DB2 バージョン 10.1 インフォメーション・センターが既にインストール済みである必要があります。詳しくは、「*DB2 サーバー機能 インストール*」の『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール』のトピックを参照してください。インフォメーション・センターのインストールに適用されるすべての前提条件と制約事項は、インフォメーション・センターの更新にも適用されます。

このタスクについて

既存の DB2 インフォメーション・センターは、自動で更新することも手動で更新することもできます。

- 自動更新は、既存のインフォメーション・センターのフィーチャーと言語を更新します。自動更新を使用すると、手動更新と比べて、更新中にインフォメーショ

ン・センターが使用できなくなる時間が短くなるというメリットがあります。さらに、自動更新は、定期的に実行する他のバッチ・ジョブの一部として実行されるように設定することができます。

- 手動更新は、既存のインフォメーション・センターのフィーチャーと言語の更新に使用できます。自動更新は更新処理中のダウン時間を減らすことができますが、フィーチャーまたは言語を追加する場合は手動処理を使用する必要があります。例えば、ローカルのインフォメーション・センターが最初は英語とフランス語でインストールされており、その後ドイツ語もインストールすることにした場合、手動更新でドイツ語をインストールし、同時に、既存のインフォメーション・センターのフィーチャーおよび言語を更新できます。しかし、手動更新ではインフォメーション・センターを手動で停止、更新、再始動する必要があります。更新処理の間はずっと、インフォメーション・センターは使用できなくなります。自動更新処理では、インフォメーション・センターは、更新を行った後に、インフォメーション・センターを再始動するための停止が発生するだけで済みます。

このトピックでは、自動更新のプロセスを詳しく説明しています。手動更新の手順については、『コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新』のトピックを参照してください。

手順

コンピューターまたはイントラネット・サーバーにインストールされている DB2 インフォメーション・センターを自動更新する手順を以下に示します。

1. Linux オペレーティング・システムの場合、次のようにします。
   a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`/opt/ibm/db2ic/V10.1` ディレクトリーにインストールされています。
   b. インストール・ディレクトリーから `doc/bin` ディレクトリーにナビゲートします。
   c. 次のように `update-ic` スクリプトを実行します。

      ```
      update-ic
      ```
2. Windows オペレーティング・システムの場合、次のようにします。
   a. コマンド・ウィンドウを開きます。
   b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、`<Program Files>¥IBM¥DB2 Information Center¥バージョン 10.1` ディレクトリーにインストールされています (`<Program Files>` は「Program Files」ディレクトリーのロケーション)。
   c. インストール・ディレクトリーから `doc¥bin` ディレクトリーにナビゲートします。
   d. 次のように `update-ic.bat` ファイルを実行します。

      ```
      update-ic.bat
      ```

タスクの結果

DB2 インフォメーション・センターが自動的に再始動します。更新が入手可能な場合、インフォメーション・センターに、更新された新しいトピックが表示されます。インフォメーション・センターの更新が入手可能でなかった場合、メッセージがログに追加されます。ログ・ファイルは、doc¥eclipse¥configuration ディレクトリーにあります。ログ・ファイル名はランダムに生成された名前です。例えば、1239053440785.log のようになります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

このタスクについて

ローカルにインストールされた *DB2 インフォメーション・センター* を手動で更新するには、以下のことを行う必要があります。

1. コンピューター上の *DB2 インフォメーション・センター* を停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。DB2 インフォメーション・センターのワークステーション・バージョンは、常にスタンドアロン・モードで実行されます。を参照してください。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールしなければならない更新がある場合は、「更新」機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに *DB2 インフォメーション・センター* の更新をインストールする必要がある場合、インターネットに接続されていて *DB2 インフォメーション・センター* がインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングしてください。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージを入手します。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の *DB2 インフォメーション・センター* を再開します。

注: Windows 2008、Windows Vista (およびそれ以上) では、このセクションの後の部分でリストされているコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを開くには、ショートカットを右クリックしてから、**「管理者として実行」**を選択します。

手順

コンピューターまたはイントラネット・サーバーにインストール済みの *DB2* インフォメーション・センター を更新するには、以下のようにします。

1. *DB2* インフォメーション・センター を停止します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**停止**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv10 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、*Program_Files*¥IBM¥DB2 Information Center¥バージョン 10.1 ディレクトリーにインストールされています (*Program_Files* は Program Files ディレクトリーのロケーション)。
     c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。
     d. 次のように help_start.bat ファイルを実行します。

        ```
        help_start.bat
        ```
   - Linux の場合:
     a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は、/opt/ibm/db2ic/V10.1 ディレクトリーにインストールされています。
     b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
     c. 次のように help_start スクリプトを実行します。

        ```
        help_start
        ```

   システムのデフォルト Web ブラウザーが開き、スタンドアロンのインフォメーション・センターが表示されます。
3. 「**更新**」ボタン () をクリックします。(ブラウザーで JavaScript が有効になっている必要があります。) インフォメーション・センターの右側のパネルで、「**更新の検索**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。
6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

```
help_end.bat
```

注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して停止しないでください。

- Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

```
help_end
```

注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを停止しないでください。

7. *DB2 インフォメーション・センター* を再開します。
   - Windows では、「**スタート**」 > 「**コントロール パネル**」 > 「**管理ツール**」 > 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

```
/etc/init.d/db2icdv10 start
```

タスクの結果

更新された *DB2 インフォメーション・センター* に、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 データベース製品のさまざまな機能について学習するための支援となります。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド*」の『**pureXML®**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

DB2 トラブルシューティング情報

DB2 データベース製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 の資料

トラブルシューティング情報は、「*問題判別およびデータベース・パフォーマンスのチューニング*」または *DB2 インフォメーション・センター* の『データベースの基本』セクションにあります。ここには、以下の情報が記載されています。

- DB2 診断ツールおよびユーティリティーを使用した、問題の切り分け方法および識別方法に関する情報。
- 最も一般的な問題のうち、いくつかの解決方法。
- DB2 データベース製品で発生する可能性のある、その他の問題の解決に役立つアドバイス。

IBM サポート・ポータル

現在問題が発生していて、考えられる原因とソリューションを見つけるには、IBM サポート・ポータルを参照してください。 Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

IBM サポート・ポータル (http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows) にアクセスしてください。

ご利用条件

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

IBM の商標: IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

付録 B. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。IBM 以外の製品に関する情報は、本書の最初の発行時点で入手可能な情報に基づいており、変更される場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。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。従って 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。サンプル・プログラムは、現存するままの状態で提供されるものであり、いかなる種類の保証も提供されません。 IBM は、これらのサンプル・プログラムの使用から生ずるいかなる損害に対しても責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。
- UNIX は The Open Group の米国およびその他の国における登録商標です。
- インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Celeron、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft、 Windows、 Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

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。

[カ行]

[サ行]

[タ行]

[ナ行]

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Printed in Japan

SA88-4677-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻