IBM® DB2 Universal Database™

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

バージョン 8.2

SC88-9160-01
(英文原典：SC09-4850-01)

IBM® DB2 Universal Database™

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

バージョン 8.2

SC88-9160-01
(英文原典：SC09-4850-01)

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、『*特記事項*』に記載されている情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC09–4850–01
IBM® DB2 Universal Database™
Call Level Interface Guide and Reference, Volume 2
Version 8.2

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.8

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

第 1 章 DB2 CLI の関数

この章では、各 DB2 CLI 関数を説明します。関数サマリーでは関数をカテゴリー順に示します。関数リストでは各関数を詳しく説明します。

CLI と ODBC 関数のサマリー

ODBC 列の **Depr** は、 ODBC 内で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 - SQL/CLI 9075-3 仕様に定義されている。
- SQL3 - 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る。

表 1. カテゴリー別の DB2 CLI 関数リスト

タスク関数名	ODBC 3.0	SQL/CLI	サポートされる DB2 CLI の最初のバージョン	目的
データ・ソースへの接続				
SQLAllocConnect()	Depr	95	V 1.1	接続ハンドルを獲得します。
SQLAllocEnv()	Depr	95	V 1.1	環境ハンドルを獲得します。 1 つまたは複数の接続に 1 つの環境ハンドルが使用されます。
SQLAllocHandle()	コア	95	V 5	ハンドルを獲得します。
SQLBrowseConnect()	レベル 1	95	V 5	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	V 1.1	データ・ソース、ユーザー ID、およびパスワードによって特定のドライバーに接続します。
SQLDriverConnect()	コア	SQL3	V 2.1 [a]	接続ストリングで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	なし	なし	DB2 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	V 5	接続属性を設定します。
SQLSetConnectOption()	Depr	95	V 2.1	接続属性を設定します。

表 1. カテゴリー別の DB2 CLI 関数リスト (続き)

タスク関数名	ODBC 3.0	SQL/CLI	サポートされる DB2 CLI の最初のバージョン	目的
SQLSetConnection()	なし	SQL3	V 2.1	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 DB2 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。
DataLink 関数				
SQLBuildDataLink()	いいえ	はい	V 5.2	DATALINK 値を作成します。
SQLGetDataLinkAttr()	いいえ	はい	V 5.2	DataLink 属性値を取得します。
ドライバーおよびデータ・ソースに関する情報の獲得				
SQLDataSources()	レベル 2	95	V 1.1	使用可能なデータ・ソースのリストを返します。
SQLGetInfo()	コア	95	V 1.1	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	V 1.1	サポートされているドライバー関数を返します。
SQLGetTypeInfo()	コア	95	V 1.1	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				
SQLSetEnvAttr()	コア	95	V 2.1	環境オプションを設定します。
SQLGetEnvAttr()	コア	95	V 2.1	環境オプションの値を返します。
SQLGetConnectAttr()	レベル 1	95	V 5	接続オプションの値を返します。
SQLGetConnectOption()	Depr	95	V 2.1 [a]	接続オプションの値を返します。
SQLSetStmtAttr()	コア	95	V 5	ステートメント属性を設定します。
SQLSetStmtOption()	Depr	95	V 2.1 [a]	ステートメント・オプションを設定します。
SQLGetStmtAttr()	コア	95	V 5	ステートメント属性の値を返します。
SQLGetStmtOption()	Depr	95	V 2.1 [a]	ステートメント・オプションの値を返します。
SQL 要求の準備				
SQLAllocStmt()	Depr	95	V 1.1	ステートメント・ハンドルを割り振ります。
SQLPrepare()	コア	95	V 1.1	後で実行するための SQL ステートメントを準備します。
SQLExtendedPrepare()	いいえ	いいえ	V 6	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()	いいえ	いいえ	V 6	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()	レベル 1	95 [b]	V 2.1	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)。

表 1. カテゴリー別の DB2 CLI 関数リスト (続き)

タスク関数名	ODBC 3.0	SQL/CLI	サポートされる DB2 CLI の最初のバージョン	目的
SQLSetParam()	Depr	なし	V 1.1	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	なし	V 2.1	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	V 1.1	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	V 1.1	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	V 5	パラメーター・マーカーの記述を返します。
SQLExecute()	コア	95	V 1.1	準備済みステートメントを実行します。
SQLExecDirect()	コア	95	V 1.1	ステートメントを実行します。
SQLNativeSql()	レベル 2	95	V 2.1 [a]	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()	レベル 2	95	V 2.1 [a]	ステートメント内のパラメーターの数を返します。
SQLParamData()	レベル 1	95	V 2.1 [a]	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。(長いデータ値の場合に便利です。)
SQLPutData()	コア	95	V 2.1 [a]	パラメーターのデータ値の一部または全部を送ります。 (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				
SQLRowCount()	コア	95	V 1.1	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	V 1.1	結果セット内の列の数を返します。
SQLDescribeCol()	コア	95	V 1.1	結果セット内の列について記述します。
SQLColAttribute()	コア	はい	V 5	結果セット内の列の属性を記述します。
SQLColAttributes()	Depr	はい	V 1.1	結果セット内の列の属性を記述します。
SQLColumnPrivileges()	レベル 2	95	V 2.1	表の列に関連した特権を獲得します。
SQLSetColAttributes()	なし	なし	V 2.1	結果セット内の列の属性を設定します。
SQLBindCol()	コア	95	V 1.1	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	V 1.1	結果行を返します。
SQLFetchScroll()	コア	95	V 5	結果行から行セットを返します。
SQLExtendedFetch()	Depr	95	V 2.1	複数の結果行を返します。

表 1. カテゴリー別の DB2 CLI 関数リスト (続き)

タスク関数名	ODBC 3.0	SQL/CLI	サポートされる DB2 CLI の最初のバージョン	目的
SQLGetData()	コア	95	V 1.1	結果セットの 1 行の 1 列の一部または全部を返します。 (長いデータ値の場合に便利です。)
SQLMoreResults()	レベル 1	SQL3	V 2.1 [a]	使用可能な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開始します。
SQLNextResult()	なし	はい	V7.1	SQLNextResult では、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	V 1.1	追加のエラー情報または状況情報を返します。
SQLGetDiagField()	コア	95	V 5	診断データのフィールドを獲得します。
SQLGetDiagRec()	コア	95	V 5	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	V 5	行セット内のカーソル位置を設定します。
SQLGetSQLCA()	なし	なし	V 2.1	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	なし	V 6	ブックマークを使用して大量の追加、更新、削除、および取り出しを実行します。
記述子				
SQLCopyDesc()	コア	95	V 5	記述子情報をハンドル間でコピーします。
SQLGetDescField()	コア	95	V 5	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	V 5	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	V 5	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	V 5	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()	なし	なし	V 2.1	LOB ファイル参照を LOB 列に関連付けます。
SQLBindFileToParam()	なし	なし	V 2.1	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。
SQLGetLength()	なし	SQL3	V 2.1	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()	なし	SQL3	V 2.1	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。

表 1. カテゴリー別の DB2 CLI 関数リスト (続き)

タスク関数名	ODBC 3.0	SQL/CLI	サポートされる DB2 CLI の最初のバージョン	目的
SQLGetSubString()	なし	SQL3	V 2.1	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 ロケーターで参照される)。
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	V 2.1 [a]	指定した表の中の列名のリストを返します。
SQLForeignKeys()	Lvl 2	SQL3	V 2.1	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	V 2.1	表の主キーを構成する列名のリストを返します。
SQLProcedureColumns()	Lvl 2	なし	V 2.1	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	なし	V 2.1	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	V 2.1 [a]	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	V 2.1 [a]	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	V 2.1	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	V 2.1 [a]	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	V 1.1	ハンドル・リソースを解放します。
SQLFreeStmt()	コア	95	V 1.1	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	V 1.1	SQL ステートメントを取り消します。
SQLTransact()	Depr	なし	V 1.1	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	V 5	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	V 1.1	接続をクローズします。
SQLEndTran()	コア	95	V 5	接続のトランザクションを終了します。
SQLFreeConnect()	Depr	95	V 1.1	接続ハンドルを解放します。

表 1. カテゴリー別の DB2 CLI 関数リスト (続き)

タスク関数名	ODBC 3.0	SQL/CLI	サポートされる DB2 CLI の最初のバージョン	目的
SQLFreeEnv()	Depr	95	V 1.1	環境ハンドルを解放します。

注:

[a] 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 でも利用可能でした。

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

ODBC 関数:

- 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。

関連概念:

- 「*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*」の『CLI の紹介』
- 「*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*」の『CLI での初期化と終了の概説』
- 「*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*」の『CLI でのトランザクション処理の概説』

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

使用すべきでない:

注:

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。

DB2 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 DB2 CLI プログラムでご使用になることをお勧めします。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「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」の『CLI でのハンドル』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』

SQLAllocEnv 関数 (CLI) - 環境ハンドルの割り振り

使用すべきでない:

注:

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 DB2 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、DB2 CLI プログラムでは SQLAllocHandle() をご使用になることをお勧めします。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「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」の『CLI でのハンドル』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』

SQLAllocHandle 関数 (CLI) - ハンドルの割り振り

目的:

仕様	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLAllocHandle() は、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る汎用関数です。

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。

構文:

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

関数引き数:

表 2. SQLAllocHandle 引き数

データ・タイプ	引き数	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、 DB2 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、出力引き数が 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* 引き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 引き数は SQL_HANDLE_STMT または SQL_HANDLE_DESC でしたが、 *InputHandle* 引き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。 DB2 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* 引き数が 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* 引き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* 引き数は、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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「*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*」の『CLI アプリケーションの初期設定』
- 「*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*」の『CLI アプリケーションでのステートメント・ハンドルの割り振り』

関連資料:

- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 319 ページの『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『clihandl.c -- How to allocate and free handles』

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

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』

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

目的:

仕様	DB2 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 (バインドする) 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 */
               SQLINTEGER       BufferLength,             /* dbValueMax */
               SQLINTEGER       *StrLen_or_IndPtr);       /* *pcbValue */
```

関数引き数:

表 4. SQLBindCol 引き数

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

表 4. SQLBindCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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。
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_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* 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLPOINTER	*TargetValuePtr*	入出力 (据え置き)	取り出しが起きたときに、 DB2 CLI が列単位または行単位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* 引き数 SQL_REFRESH を使用する関数 `SQLFetch()`、`SQLFetchScroll()`、`SQLSetPos()` の呼び出しか、または *Operation* 引き数 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。
SQLINTEGER	*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、この引き数は無視されます。

表 4. SQLBindCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	*StrLen_or_IndPtr*	入出力 (据え置き)	DB2 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* 引き数 SQL_REFRESH を使用する関数 SQLFetch()、SQLFetchScroll()、SQLSetPos() の呼び出しか、または *Operation* 引き数 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() はこの引き数に 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、後述の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、結果セット列が取り出されるまでは、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。たとえば、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、 DB2 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。

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取り出し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、すでに取り出したデータには適用されず、次回の取り出しから使用されます。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、 DB2 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可能な実際のサイズの *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、値が取り出されたときに 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* バッファーに取り出されるデータは、 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 NULL 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 DB2 CLI に GRAPHIC ストリングを強制的に NULL 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関連付けを解除する必要があります。

引き数のマッピング

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* 引き数の値が 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 つまたは複数の 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 引き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。取り出しがその列について 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、取り出し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、取り出し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* 引き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* 引き数の値が NULL ポインターで、 *ColumnNumber* 引き数の値が 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、取り出す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	引き数 *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。
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。

表 5. SQLBindCol SQLSTATEs (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。	引き数 *BufferLength* に指定された値は 1 より小さく、引き数 *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。	DB2 CLI は引き数 *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: 取り出し時に、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 **HY**C00 が返されます。

例:

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

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでの LOB ロケーター』
- 「*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*」の『CLI アプリケーションの記述子』

関連タスク:

- 「*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*」の『CLI アプリケーションでの照会結果の取り出し』
- 「*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*」の『CLI アプリケーションでの列方向バインドを使用した配列データの取り出し』
- 「*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*」の『CLI アプリケーションでの行方向バインドを使用した配列データの取り出し』

関連資料:

- 「*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*」の『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』
- 「*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*」の『CLI アプリケーション用の C データ・タイプ』
- 19 ページの『SQLBindFileToCol 関数 (CLI) - LOB 列への LOB ファイル参照のバインド』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』

- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 322 ページの『SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定』
- 50 ページの『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 「*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*」の『CLI 関数戻りコード』
- 「*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*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

関連サンプル:

- 『tbinfo.c -- How to get information about tables from the system catalog tables』
- 『tut_mod.c -- How to modify table data』
- 『tut_read.c -- How to read data from tables』

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

目的:

仕様:	DB2 CLI 2.1		

SQLBindFileToCol() を使用するのは、結果セット内の LOB 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する場合です。この処理により、ステートメント・ハンドル用に各行が取り出された時点で、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。

LOB ファイル参照引き数 (ファイル名、ファイル名の長さ、ファイル参照オプション) は、アプリケーションの環境内 (クライアント上) のファイルを参照します。各行を取り出す前に、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、各行の取り出し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);
```

関数引き数:

表 6. SQLBindFileToCol 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
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)	次回の取り出しの時に *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)	次回の取り出しの時に *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 NULL 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)	次回の取り出しの時に *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、行が取り出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。

FileName、 *FileNameLength*、および *FileOptions* は各取り出し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。取り出しの際に、SQLBindFileToCol() の据え置き入力引き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* 出力引き数は、取り出し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、 DB2 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。	引き数 *icol* に指定された値は、1 より小さい値でした。 引き数 *icol* に指定された値が、データ・ソースでサポートされ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。

表 7. SQLBindFileToCol 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。
HY009	引き数の値が無効です。	*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。	引き数 *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* 出力引き数を調べて、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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのラージ・オブジェクトの使用』

関連タスク:

- 「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」の『CLI アプリケーションでの照会結果の取り出し』

関連資料:

- 11 ページの『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』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 27 ページの『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』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』

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

目的:

仕様	DB2 CLI 2.1		

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 列に直接転送できるようになります。

LOB ファイル参照引き数 (ファイル名、ファイル名の長さ、ファイル参照オプション) は、アプリケーションの環境内 (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);
```

関数引き数:

表 8. SQLBindFileToParam 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
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
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。 この引き数を 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 NULL 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 引き数 (続き)

データ・タイプ	引き数	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、 DB2 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	引き数の値が無効です。	*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。	入力引き数 *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。	引き数 *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* 出力引き数を調べて、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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「*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*」の『CLI アプリケーションでのパラメーター・マーカーのバインディング』

関連資料:

- 「*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*」の『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』
- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』

- 271 ページの『SQLParamOptions 関数 (CLI) - パラメーターの入力配列の指定』
- 27 ページの『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』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』

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

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 2.0	

SQLBindParameter() は、 SQL ステートメント内のパラメーター・マーカーを以下のいずれかに関連付けたりバインドしたりするのに使用します。

- 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)。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 */
              SQLUINTEGER       ColumnSize,          /* cbColDef */
              SQLSMALLINT       DecimalDigits,       /* ibScale */
              SQLPOINTER        ParameterValuePtr,   /* rgbValue */
              SQLINTEGER        BufferLength,        /* cbValueMax */
              SQLINTEGER        *StrLen_or_IndPtr);  /* pcbValue */
```

関数引き数:

表 10. SQLBindParameter 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
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。
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、この引き数に設定されます。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 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*ValueType*	入力	パラメーターの C データ・タイプ。以下のタイプがサポートされます。 • 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_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 引き数 (続き)

データ・タイプ	引き数	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_LONG • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SHORT • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR **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 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUINTEGER	*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。 • バイナリーまたは単一バイト文字ストリング (たとえば、 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 (たとえば、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。 • それ以外の場合は、この引き数は無視されます。
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 (たとえば、 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 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLPOINTER	*ParameterValuePtr*	入力 (据え置き) または出力 (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 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*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、この引き数は無視されます。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、 NULL 終了します (NULL 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 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	*StrLen_or_IndPtr*	入力 (据え置き) または出力 (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 NULL 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 NULL 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。

表 10. SQLBindParameter 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER *	StrLen_or_IndPtr *(cont)*	入力 (据え置き) または出力 (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。 これが出力パラメーターである (*InputOutputType* が SQL_PARAM_OUTPUT に設定されている) 場合: これは、ストアード・プロシージャー呼び出し (CALL) の出力パラメーターまたは戻り値でなければならず、ストアード・プロシージャー実行後に以下のいずれかを指します。 • *ParameterValuePtr* 内に返すために使用できるバイト数 (NULL 終止符文字を除く)。 • SQL_NULL_DATA • SQL_NO_TOTAL (返すために使用できるバイト数を判別できない場合)。

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 文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 引き数は、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・プロシージャー引き数の規則では、すべてのプロシージャー引き数は入出力であることが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、DB2 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、 DB2 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* 引き数が両方とも NULL ポインターである場合、 DB2 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、 DB2 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* は据え置き引き数です。 *InputOutputType* が SQL_PARAM_INPUT または SQL_PARAM_INPUT_OUTPUT に設定されている場合、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() 呼び出しが SQLBindParameter() 呼び出しと同じプロシージャー有効範囲に保持されているか、あるいは、これらの保管ロケーションが動的に割り振られているか静的またはグローバルに宣言されているかのいずれかです。

  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 引き数:

ParameterValuePtr と *StrLen_or_IndPtr* は据え置き引き数であるので、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 つまたは複数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 DB2 CLI に通知します。また、 *ParameterValuePtr* 入力引き数を 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、 DB2 CLI のパフォーマンスを若干向上させることができます。たとえば、次のようになります。

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

BufferLength 引き数:

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* 引き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* 引き数は無視されます。

ColumnSize 引き数:

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、 DB2 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、 DB2 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、ステートメントの実行時に判別される入力値の実際のデータ長を使用します。 DB2 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、 DB2 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、 DB2 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。ランタイムに経費のかさむデータ長のチェックが DB2 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()` 内で引き数を使用します。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* 引き数の値が 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* 引き数の値が 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 つまたは複数の 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* 引き数によって識別されるデータ値から *ParameterType* 引き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。 (たとえば、SQL_C_DATE から SQL_DOUBLE への変換は無意味です。)
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。	引き数 *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	引き数 *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	引き数の値が無効です。	引き数 *ParameterValuePtr* は NULL ポインターで、引き数 *StrLen_or_IndPtr* も NULL ポインターですが、 *InputOutputType* 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。	引き数 *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。	引き数 *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き数 *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_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 6 より大きい値でした。
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。

表 11. SQLBindParameter SQLSTATE (続き)

SQLSTATE	説明	解説
HYC00	ドライバーが使用できません。	DB2 CLI またはデータ・ソースが、引き数 *ValueType* に指定された値と引き数 *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。 引き数 *ParameterType* に指定された値が、 DB2 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:

DB2 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・プロシージャー引き数ではデフォルト値はサポートされないので、*StrLen_or_IndPtr* 引き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* 引き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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのパラメーター・マーカー・バインディング』

関連タスク:

- 「*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*」の『CLI アプリケーションからのストアード・プロシージャーの呼び出し』

関連資料:

- 23 ページの『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』
- 268 ページの『SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得』
- 271 ページの『SQLParamOptions 関数 (CLI) - パラメーターの入力配列の指定』
- 293 ページの『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『tbread.c -- How to read data from tables』

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

目的:

仕様:	DB2 CLI 5.0	ODBC 1	

SQLBrowseConnect() は、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。 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 12. *SQLBrowseConnect* 引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLCHAR *	*InConnectionString*	入力	要求接続ストリングをブラウズします (44 ページの『InConnectionString 引き数』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 (45 ページの『OutConnectionString 引き数』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。
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。 DB2 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* 引き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、DB2 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()` 用の引き数として 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、たと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。 DB2 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 引き数:

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

ここで、

- 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 DB2 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。 DB2 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、 DB2 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、 DB2 CLI は、最初に現れたキーワードの方を使用します。

OutConnectionString 引き数:

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、DB2 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

ここで、

- 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 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 Universal Database 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 DB2 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 DB2 CLI によって返されます。たとえば、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。たとえば、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。 DB2 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。	DB2 CLI は、`SQLSetConnectAttr()` 内の *ValuePtr* 引き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。	DB2 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、 DB2 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。

表 13. SQLBrowseConnect SQLSTATE (続き)

SQLSTATE	説明	解説
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。
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。	引き数 *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。 引き数 *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: %s¥n", outStr);

  /* 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 %s.¥n", db1Alias);
  }
}
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 83 ページの『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 100 ページの『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 103 ページの『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbcongui.c -- How to connect to a database with a graphical user interface (GUI)』

SQLBuildDataLink 関数 (CLI) - DATALINK 値の作成

目的:

仕様:	DB2 CLI 5.2		ISO CLI

SQLBuildDataLink() は、入力引き数から作成された DATALINK 値を戻します。

構文:

```
SQLRETURN   SQLBuildDataLink (
               SQLHSTMT        StatementHandle,     /* hStmt */
               SQLCHAR         *LinkType,           /* *pszLinkType */
               SQLINTEGER      LinkTypeLength,      /* cbLinkType */
               SQLCHAR         *DataLocation,       /* *pszDataLocation */
               SQLINTEGER      DataLocationLength,  /* cbDataLocation */
               SQLCHAR         *Comment,            /* *pszComment */
               SQLINTEGER      CommentLength,       /* cbComment */
               SQLCHAR         *DataLinkValue,      /* *pDataLink */
               SQLINTEGER      BufferLength,        /* cbDataLinkMax */
               SQLINTEGER      *StringLengthPtr);   /* *pcbDataLink */
```

関数引き数:

表 14. SQLBuildDataLink 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	診断報告専用です。

表 14. SQLBuildDataLink 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*LinkType*	入力	常に SQL_DATALINK_URL に設定されます。
SQLINTEGER	*LinkTypeLength*	入力	*LinkType* 値の長さ。
SQLCHAR *	*DataLocation*	入力	割り当てる完全な URL 値。
SQLINTEGER	*DataLocationLength*	入力	*DataLocation* 値の長さ。
SQLCHAR *	*Comment*	入力	割り当てるコメントがある場合、そのコメント。
SQLINTEGER	*CommentLength*	入力	*Comment* 値の長さ。
SQLCHAR *	*DataLinkValue*	出力	関数によって作成される DATALINK 値。
SQLINTEGER	*BufferLength*	入力	*DataLinkValue* バッファーの長さ。
SQLINTEGER *	*StringLengthPtr*	出力	**DataLinkValue* 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。 *DataLinkValue* が NULL ポインターである場合、長さは戻されません。戻りに使用できるバイト数が BufferLength から NULL 終止符文字の長さを引いた長さより大きい場合、 SQLSTATE 01004 が戻されます。その場合、DATALINK 値を続けて使用すると障害が起こることがあります。

使用法:

関数は DATALINK 値の作成に使用されます。 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。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 15. SQLBuildDataLink() SQLSTATE

SQLSTATE	説明	解説
01000	警告。	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。
01004	データが切り捨てられました。	**DataLinkValue* 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。)
HY000	一般的なエラーです。	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 15. SQLBuildDataLink() 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。
HY090	ストリングまたはバッファーの長さが無効です。	引き数の 1 つ (*LinkTypeLength*、*DataLocationLength*、または *CommentLength*) に指定された値が 0 より小さく SQL_NTS に等しくないか、または *BufferLength* が 0 より小さい値です。

制限:

なし。

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 177 ページの『SQLGetDataLinkAttr 関数 (CLI) - DataLink 属性値の取得』
- 「*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*」の『CLI 関数戻りコード』

SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し

目的:

仕様:	DB2 CLI 6.0	ODBC 3.0	

キー・セット主導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取り出す

構文:

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

関数引き数:

表 16. SQLBulkOperations の引き数

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

表 16. SQLBulkOperations の引き数 (続き)

データ・タイプ	引き数	使用法	説明
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・セット主導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取り出す

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 (たとえば、 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* 引き数として 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* 引き数として 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 引き数が 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 17. 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* 引き数は SQL_FETCH_BY_BOOKMARK だったので、1 つ以上の列別に戻されたデータ・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* 引き数は 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 つまたは複数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。)
07006	制限付きデータ・タイプ属性違反。	*Operation* 引き数は 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* 引き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* 引き数は 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	引き数 *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。	引き数 *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。

表 17. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
22003	範囲外の数値。	*Operation* 引き数は 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。 引き数 *Operation* は SQL_FETCH_BY_BOOKMARK で、 1 つまたは複数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* 引き数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK で、結果セットの列への日付またはタイム・スタンプ値の割り当てで、年、月、または日フィールドの範囲が超過しました。 引き数 *Operation* は SQL_FETCH_BY_BOOKMARK で、 1 つまたは複数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の範囲が超過し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* 引き数は SQL_ADD または SQL_UPDATE_BY_BOOKMARK で、結果セットの列に送られるデータの日時計算で、結果の日時フィールド (年、月、日、時、分、または秒フィールド) が許容範囲を超えたか、または、グレゴリオ歴に基づく日時の法則に対して無効な値になりました。 *Operation* 引き数は SQL_FETCH_BY_BOOKMARK で、結果セットから検索されるデータの日時計算で、結果の日時フィールド (年、月、日、時、分、または秒フィールド) が許容範囲を超えたか、または、グレゴリオ歴に基づく日時の法則に対して無効な値になりました。
22018	キャスト指定の文字値が無効。	*Operation* 引き数は 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。 引き数 *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* 引き数は 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* 引き数は 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* 引き数は 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。

表 17. SQLBulkOperations SQLSTATE (続き)

SQLSTATE	説明	解説
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。	DB2 CLI は *Operation* 引き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* 引き数は 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。
HY008	操作が取り消されました。	*StatementHandle* で非同期処理が使用可能になりました。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。	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 17. SQLBulkOperations 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* 引き数は 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* 引き数は 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* 引き数は 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。)
HY092	無効な属性 ID。	*Operation* 引き数に指定された値は無効でした。 *Operation* 引き数は 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* 引き数は 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	オプショナル機能はインプリメントされませんでした。	DB2 CLI またはデータ・ソースは、 *Operation* 引き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* 引き数として 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。

関連概念:

- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションのカーソル』
- 「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」の『CLI アプリケーションでのバルク挿入およびバルク更新用の長いデータ』

関連タスク:

- 「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」の『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの挿入』
- 「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」の『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの検索』
- 「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」の『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの削除』
- 「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」の『CLI アプリケーションでの SQLBulkOperations() を使用したブックマークによるバルク・データの更新』

関連資料:

- 11 ページの『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』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 201 ページの『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 「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」の『CLI 関数戻りコード』

SQLCancel 関数 (CLI) - ステートメントの取り消し

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

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・シーケンスを早期終了することができます。

また SQLCancel() 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。

構文:

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

関数引き数:

表 18. SQLCancel 引き数

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

使用法:

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。

注: DB2 Universal Database for z/OS™ および OS/390®、バージョン 7 以前、または DB2 for iSeries™ など、ネイティブ割り込みサポートのないサーバーに対して SQLCancel() を呼び出すには、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。

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 DB2 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、DB2 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 19. 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、 DB2 CLI によって生成された一時ファイルがクローズし、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);
```

関連概念:

- 「*DB2 Connect ユーザーズ・ガイド*」の『DCS ディレクトリーの値』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『割り込み要求の処理』
- 「*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*」の『CLI アプリケーションのカーソル』
- 「*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*」の『CLI アプリケーションでのラージ・オブジェクトの使用』
- 「*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*」の『マルチスレッド CLI アプリケーション』
- 「*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*」の『CLI アプリケーションでのバルク挿入およびバルク更新用の長いデータ』

関連タスク:

- 「*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*」の『CLI アプリケーションでの長形式データ操作のための実行時パラメーター値の指定』

関連資料:

- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 268 ページの『SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』

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

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLCloseCursor() は、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。

構文:

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

関数引き数:

表 20. SQLCloseCursor 引き数

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

使用法:

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 DB2 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 21. 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、DB2 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。 *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。

制限:

なし。

例:

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

関連概念:

- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションのカーソル』

関連資料:

- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 257 ページの『SQLMoreResults 関数 (CLI) - さらに結果セットがあるかどうかの判別』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*SQL リファレンス 第 2 巻*」の『CLOSE ステートメント』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』
- 『udfcli.c -- How to work with different types of user-defined functions (UDFs)』

SQLColAttribute 関数 (CLI) - 列属性を戻す

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLColAttribute() は、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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 Unicode 関数 (CLI) を参照してください。

構文:

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

関数引き数:

表 22. SQLColAttribute 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
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。この引き数は、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。 この引き数内に列 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 (63 ページの表 23 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。
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* 以上であれば、*CharacterAttributePtr* 内の記述子情報は *BufferLength* バイトから NULL 終止符文字分の長さを差し引いたバイトに切り捨てられ、DB2 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、 DB2 CLI は、**CharacterAttributePtr* のサイズを 32 ビットと想定します。
SQLPOINTER	*NumericAttributePtr*	出力	フィールドが SQL_DESC_COLUMN_LENGTH のような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整数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、DB2 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、 DB2 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。下記にリストされる *FieldIdentifier* 値について `SQLColAttribute()` によって返される値は、適当な 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 DB2 CLI のバージョン、およびそれについて情報が返される引き数を、以下に示します。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。

DB2 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、 DB2 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 23. SQLColAttribute 引き数

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 DB2 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 DB2 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。

表 23. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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・タイプには SQL_FALSE が返されます。
SQL_DESC_CATALOG_NAME (DB2 CLI v2)	Character AttributePtr	DB2 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、たとえば、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。
SQL_DESC_DISTINCT_TYPE (DB2 CLI v2)	Character AttributePtr	列のユーザー定義の固有タイプ名が、*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の固有タイプではない場合、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。

表 23. *SQLColAttribute* 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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 NULL 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。
SQL_DESC_LITERAL_PREFIX (DB2 CLI v5)	Character AttributePtr	この VARCHAR(128) レコード・フィールドには、DB2 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、DB2 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。
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。

表 23. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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 10 進数を含むからです。 • 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 (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取り出し時または 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 (たとえば、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 (たとえば、SQL_VARCHAR または SQL_BLOB) である場合、最大長が返されます。

表 23. *SQLColAttribute* 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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。 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』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。
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。 DB2 CLI がこの属性を判別できないと、空ストリングが返されます。
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: どの述部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	DB2 CLI がこの属性を判別できないと、空ストリングが返されます。

表 23. SQLColAttribute 引き数 (続き)

FieldIdentifier	情報の戻り先	説明
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』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、つまり、 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。
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。

表 23. *SQLColAttribute 引き数 (続き)*

FieldIdentifier	情報の戻り先	説明
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。 • DB2 SQL データ・タイプの場合はすべて、SQL_ATTR_READWRITE_UNKNOWN が *NumericAttributePtr* に返されます。これが戻されるのは、列が更新可能かどうかを現在 DB2 CLI で確かめることはできないからです。 UNIX および Windows サーバー用の将来のバージョンの DB2 CLI では、列が更新可能かどうかを判別できるようになる予定です。 • 列をカタログ関数呼び出しから入手した場合には、SQL_ATTR_READONLY が返されます。 ODBC は以下の値 (DB2 CLI からは返されません) も定義しています。 • SQL_ATTR_WRITE

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可能な、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 24. *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。

表 24. SQLColAttribute SQLSTATE (続き)

SQLSTATE	説明	解説
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。引き数 *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。
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可能になりました。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。	引き数 *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。	引き数 *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。	引き数 *FieldIdentifier* に指定された値は、DB2 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)
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションの記述子』

関連資料:

- 「*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*」の『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』
- 11 ページの『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』
- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 93 ページの『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 427 ページの『データ・タイプ精度 (CLI) 表』
- 428 ページの『データ・タイプ・スケール (CLI) 表』
- 429 ページの『データ・タイプ長 (CLI) 表』
- 431 ページの『データ・タイプ表示 (CLI) 表』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbread.c -- How to read data from tables』
- 『utilcli.c -- Utility functions used by DB2 CLI samples』

SQLColAttributes 関数 (CLI) - 列属性の取得

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、Unicode 関数 (CLI) を参照してください。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』

関連資料:

- 61 ページの『SQLColAttribute 関数 (CLI) - 列属性を戻す』

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

目的:

仕様:	DB2 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。情報は SQL 結果セット内に返されますが、照会で生成された結果セットの処理に使用される関数と同じ関数を使用して、情報を検索することができます。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 25. SQLColumnPrivileges 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CatalogName*	入力	3 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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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* を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、 74 ページの『SQLColumnPrivileges で返される列』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、この関数の入力引き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) 照会に多くの場合マッピングされるため、それらの呼び出しの使用を少なくし、呼び出しを繰り返すのではなく結果を保管するようにしてください。

ColumnName 入力引き数は検索パターンを受け入れますが、他のどの入力引き数もこれを受け入れないことに注意してください。

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 Universal Database、DB2 for MVS/ESA、および DB2 Server for VSE & VM は、UPDATE (更新) 列特権をサポートしており、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 Universal Database、DB2 for MVS/ESA、および DB2 Server for VSE & VM 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: DB2 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() 結果セットに指定されたものと同一です。

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 26. 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。
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可能になりました。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	引き数の値が無効です。	*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。	名前の長さ引き数のうちの 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);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 76 ページの『SQLColumns 関数 (CLI) - 表の列の情報の取得』
- 353 ページの『SQLTables 関数 (CLI) - 表の情報の取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbinfo.c -- How to get information about tables from the system catalog tables』

SQLColumns 関数 (CLI) - 表の列の情報の取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLColumns() は、指定された表の中の列のリストを返します。 情報は SQL 結果セット内に返されますが、照会で作成された結果セットを取り出すのに使用される関数と同じ関数を使用して情報を検索することができます。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 27. SQLColumns 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CatalogName*	入力	3 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 分割の命名法をサポートしていない (DB2 UDB for UNIX または Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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	スキーマ名別に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	*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。
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* を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:

UNIX または Windows システムに対する照会の実行時に *CatalogName* 入力引き数 (非 NULL) を指定すると、どちらのプラットフォームも 3 分割名を使用しないので、空の結果セットが戻されます。

表および表の集まりの列に関する情報を検索するには、この関数を呼び出します。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もできます。アプリケーションは、SQLTables() 結果セットの TABLE_SCHEMA、 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する必要があります。

SQLColumns() は、TABLE_CAT、TABLE_SCHEM、TABLE_NAME、および ORDINAL_POSITION の順序で標準の結果セットを返します。 79 ページの 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* 引き数は、探索パターンを受け入れます。

この関数は、結果セット内の列に関する情報を返しません。よって SQLDescribeCol() または SQLColAttribute() 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 DB2 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() 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を少なくし、呼び出しを繰り返すのではなく結果を保管するようにしてください。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) の余地を確保しておくか、または SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続先の 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。

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。

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() への呼び出しを最適化するよう DB2 CLI/ODBC ドライバーを設定できます。

- OPTIMIZESQLCOLUMNS DB2 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なくなります。
- 列 12 REMARKS
- 列 13 COLUMN_DEF

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 の入ったスキーマの名前。

列 3 TABLE_NAME (VARCHAR(128) 非 NULL)
表、ビュー、別名、または同義語の名前。

列 4 COLUMN_NAME (VARCHAR(128) 非 NULL)
列の ID。指定された表、ビュー、別名、または同義語の列の名前。

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。これは、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、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の必要な 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、これは結果表の NUM_PREC_RADIX 列の値に基づいて、列に許可されている合計桁数または合計ビット数のいずれかです。

データ・タイプ精度の表も参照してください。

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 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・タイプ長の表も参照してください。

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・スケールの表も参照してください。

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 にはそのパラメーター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、 78 ページの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 を返します。切り捨てを行わずに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 が含まれています。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、 78 ページの 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、 79 ページの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)

文字データ・タイプ列の場合はオクテット単位の最大長が含まれています。

1 バイト文字セットの場合、これは COLUMN_SIZE と同じです。他のすべてのデータ・タイプの場合は NULL です。

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、それ以外の場合は「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。

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

診断:

表 28. 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可能になりました。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 28. SQLColumns 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。	名前の長さ引き数のうちの 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:

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

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連タスク:

- 「*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*」の『CLI アプリケーションでの照会結果の取り出し』

関連資料:

- 「*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*」の『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』
- 「*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*」の『CLI でサポートされているデータ変換』
- 72 ページの『SQLColumnPrivileges 関数 (CLI) - 表の列に関連した特権の取得』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』

- 353 ページの『SQLTables 関数 (CLI) - 表の情報の取得』
- 427 ページの『データ・タイプ精度 (CLI) 表』
- 428 ページの『データ・タイプ・スケール (CLI) 表』
- 429 ページの『データ・タイプ長 (CLI) 表』
- 「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」の『CLI 関数戻りコード』
- 「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」の『CLI/ODBC 構成キーワード (カテゴリー別)』

関連サンプル:

- 『tbinfo.c -- How to get information about tables from the system catalog tables』

SQLConnect 関数 (CLI) - データ・ソースへの接続

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLConnect() は、ターゲット・データベースに対する接続を確立します。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。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 29. SQLConnect 引き数

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

表 29. SQLConnect 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*UserName*	入力	許可名 (ユーザー ID)
SQLSMALLINT	*UserNameLength*	入力	*UserName* 引き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)
SQLSMALLINT	*AuthenticationLength*	入力	*Authentication* 引き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() の入力長さ引き数 (*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* 引き数値にはブランクを入れてはなりません。

DB2 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* 引き数がすべて NULL に設定され、そのそれぞれの長さ引き数がすべて 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。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 30. SQLConnect SQLSTATE

SQLSTATE	説明	解説
08001	データ・ソースに接続できませんでした。	DB2 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。
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。

表 30. SQLConnect SQLSTATE (続き)

SQLSTATE	説明	解説
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。	引き数 *UserName* で指定された値または引き数 *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。	引き数 *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、引き数 *ServerName* は NULL ポインターではありませんでした。 引き数 *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、引き数 *UserName* は NULL ポインターではありませんでした。 引き数 *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、引き数 *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。	引き数 *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);
```

関連概念:

- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「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」の『CLI アプリケーションの初期設定』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 90 ページの『SQLDataSources 関数 (CLI) - データ・ソースのリストの取得』
- 100 ページの『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 103 ページの『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 164 ページの『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dbconn.c -- How to connect to and disconnect from a database』
- 『dbmcon.c -- How to use multiple databases』
- 『dbmconx.c -- How to use multiple databases with embedded SQL.』
- 『spserver.c -- Definition of various types of stored procedures』

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

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLCopyDesc() は、 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。

構文:

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

関数引き数:

表 31. SQLCopyDesc 引き数

データ・タイプ	引き数	使用法	説明
SQLHDESC	*SourceDescHandle*	入力	ソース記述子ハンドル

表 31. SQLCopyDesc 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、表から取り出した行を記述する 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 32. 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、 DB2 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。
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。

表 32. SQLCopyDesc 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。
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* 引き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);
```

関連概念:

- 「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」の『CLI でのハンドル』
- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションの記述子』
- 「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」の『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 194 ページの『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 309 ページの『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』

- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』

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

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLDataSources() は、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化する必要があります。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 33. SQLDataSources 引き数

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

表 33. SQLDataSources 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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* より大きい場合、**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。 DB2 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。
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* より大きい場合、 **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、 DB2 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 34. SQLDataSources SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	引き数 *ServerName* に返されたデータ・ソース名が、引き数 *BufferLength1* に指定した値よりも長い値でした。引き数 *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。) 引き数 *Description* に返されたデータ・ソース名が、引き数 *BufferLength2* に指定した値よりも長い値でした。引き数 *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* 引き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。	引き数 *BufferLength1* に指定された値は、0 より小さい値でした。 引き数 *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。	引き数 *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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「*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*」の『CLI 環境のセットアップ』

関連資料:

- 83 ページの『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『ininfo.c -- How to get information at the instance level』

SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLDescribeCol() は、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) を返します。

この情報は、IRD のフィールドでも使用可能です。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
               SQLUINTEGER       *ColumnSizePtr,     /* pcbColDef */
               SQLSMALLINT       *DecimalDigitsPtr,  /* pibScale */
               SQLSMALLINT       *NullablePtr);      /* pfNullable */
```

関数引き数:

表 35. SQLDescribeCol 引き数

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

表 35. SQLDescribeCol 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUINTEGER *	*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_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』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* 引き数を 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。

ポインター引き数に NULL ポインターを指定すると、 DB2 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 36. SQLDescribeCol SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	引き数 * *ColumnName* 内に返された列名は、引き数 *BufferLength* 内で指定された値より長い値でした。引き数 * *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。引き数 *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可能になりました。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。
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、引き数 *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、DB2 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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションの記述子』

関連資料:

- 「*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*」の『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』
- 61 ページの『SQLColAttribute 関数 (CLI) - 列属性を戻す』
- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 266 ページの『SQLNumResultCols 関数 (CLI) - 結果列の数の取得』
- 272 ページの『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbread.c -- How to read data from tables』

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

目的:

仕様:	DB2 CLI 5.0	ODBC 1.0	ISO CLI

SQLDescribeParam() は、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。この情報は、IPD のフィールドでも使用可能です。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 */
               SQLUINTEGER       *ParameterSizePtr,   /* pcbParamDef */
               SQLSMALLINT       *DecimalDigitsPtr,   /* pibScale */
               SQLSMALLINT       *NullablePtr);       /* pfNullable */
```

関数引き数:

表 37. SQLDescribeParam 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
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。
SQLUINTEGER *	*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・メ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。 以下のいずれかになります。 • 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。

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 38. 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。	引き数 *ParameterNumber* に指定された値は、1 より小さい値でした。 引き数 *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、DB2 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可能になりました。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 38. SQLDescribeParam SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラーです。	*StatementHandle* で SQLPrepare() または SQLExecDirect() を呼び出す前に、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()、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。

制限:

なし。

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのパラメーター・マーカー・バインディング』
- 「*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*」の『CLI アプリケーションの記述子』

関連資料:

- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 261 ページの『SQLNumParams 関数 (CLI) - SQL ステートメント内のパラメーター数の取得』
- 272 ページの『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 281 ページの『SQLProcedureColumns 関数 (CLI) - プロシージャーの入力/出力パラメーター情報の取得』
- 27 ページの『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』
- 「*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*」の『CLI 関数戻りコード』

SQLDisconnect 関数 (CLI) - データ・ソースからの切断

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLDisconnect() は、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。

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 */
```

関数引き数:

表 39. SQLDisconnect 引き数

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

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、 DB2 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。たとえば、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 40. 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。	引き数 *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。	引き数 *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、DB2 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。	引き数 *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。
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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「*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*」の『CLI アプリケーションの終了』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 83 ページの『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 103 ページの『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』

- 109 ページの『SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了』
- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbconn.c -- How to connect to and disconnect from a database』
- 『dbmcon.c -- How to use multiple databases』
- 『dbmconx.c -- How to use multiple databases with embedded SQL.』
- 『spserver.c -- Definition of various types of stored procedures』

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

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLDriverConnect() は、SQLConnect() の代わりの関数です。両方の関数とも宛先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。

SQLConnect() でサポートされる 3 つの入力引き数 (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 DB2 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したい場合に、 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、Unicode 関数 (CLI) を参照してください。

構文:

総称

```
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 */
```

関数引き数:

表 41. SQLDriverConnect 引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル
SQLHWND	*WindowHandle*	入力	ウィンドウ・ハンドル。 Windows プラットフォームでは、これは親ウィンドウ・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。
SQLCHAR *	*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。
SQLCHAR *	*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	DB2 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 引き数

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
```

ここで、

- 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 DB2 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。 DB2 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、 DB2 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、 DB2 CLI は、最初に現れたキーワードの方を使用します。

OutConnectionString 引き数

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]

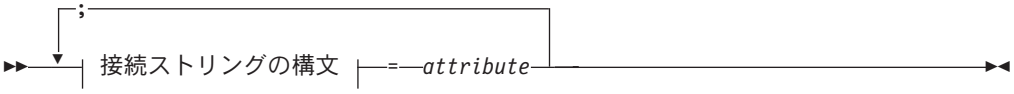
attribute-value ::= {attribute-value-list} | ?
(中括弧はリテラルであり、DB2 CLI によって返されます。)
attribute-value-list ::= character-string [:localized-character
string] | character-string [:localized-character string], attribute-value-list

ここで、

- 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、 DB2 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。



接続ストリングの構文

上記の各キーワードには、以下のような属性があります。

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 DB2 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* 引き数には 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、それらのキーワードとそれらに対応する値は、接続ストリング内の DB2 CLI に渡される情報を追加するのに使用されます。 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。

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 42. 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。引き数 **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を指定し、以下のうちの 1 つが発生しましたが、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。	引き数 *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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「*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*」の『CLI アプリケーションの初期設定』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 83 ページの『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbcongui.c -- How to connect to a database with a graphical user interface (GUI)』
- 『dbconn.c -- How to connect to and disconnect from a database』

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

目的:

仕様:	DB2 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 */
```

関数引き数:

表 43. SQLEndTran 引き数

データ・タイプ	引き数	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	*Handle*	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。詳細については、『使用法』のセクションを参照してください。
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、 DB2 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 つまたは複数の接続に

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 を使用する必要があります。

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* を指定して割り振り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、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、 SQL_SUCCESS が返され (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)、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。

DB2 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、 DB2 CLI ドライバーは SQLEndTran() ステートメントをサーバーに渡しません。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 44. 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でした。
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、要求された **COMMIT** または **ROLLBACK** が失敗する前に行われたかどうかを判別できません。
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。	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。
HY012	トランザクション・コードが無効です。	引き数 *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でした。
HY092	オプション・タイプが範囲外です。	引き数 *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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』

関連タスク:

- 「*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*」の『CLI アプリケーションの終了』

関連資料:

- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 194 ページの『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 201 ページの『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『tbmod.c -- How to modify table data』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』

SQLError 関数 (CLI) - エラー情報の取り出し

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 Unicode 関数 (CLI) を参照してください。

新しい関数への移行

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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでの診断の概説』

関連資料:

- 188 ページの『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 194 ページの『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*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*」の『CLI 関数戻りコード』

SQLExecDirect 関数 (CLI) - ステートメントの直接実行

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQL ステートメント内にパラメーターが存在する場合、SQLExecDirect() はそのパラメーター・マーカー変数の現行値を使って、指定された SQL ステートメントを直接実行します。ステートメントは、1 回だけ実行することができます。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 45. SQLExecDirect 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。 *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* 引き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・エスケープ文節シーケンス列が入っている場合、 DB2 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・エスケープ文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_NOSCAN_ON に SQL_ATTR_NOSCAN を設定し、 DB2 CLI がベンダー・エスケープ文節をスキャンして探索しないようにしなければなりません。

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、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、 DB2 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取り出します。

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_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 46. SQLExecDirect SQLSTATE

SQLSTATE	説明	解説
01504	UPDATE または DELETE ステートメントに、WHERE 文節がありません。	*StatementText* の UPDATE ステートメントまたは DELETE ステートメントに、 WHERE 文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、引き数 *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。	DB2 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。

表 46. SQLExecDirect 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。	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。
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 Universal Database 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き数 *fSQLType* は SQL GRAPHIC データ・タイプを表しますが、据え置き長さ引き数 (*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。たとえば、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 内に保全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。

表 46. SQLExecDirect SQLSTATE (続き)

SQLSTATE	説明	解説
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
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	**42**5xx は、 *StatementText* に含まれている SQL ステートメントの実行がこの許可 ID に許可され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許可を持っていません。
		• 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、ディスクがいっぱいになりました。
		• 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き数 *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 46. 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、他の保全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 AS/400 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 文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	引き数の値が無効です。	*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。	引き数 *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* 引き数が無効でした。
HY503	ファイル名の長さが無効です。	`SQLBindFileToParam()` からの *fileNameLength* 引き数値は 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 46. 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。たとえば、 **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);
```

関連概念:

- 「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*」の『CLI アプリケーションでのベンダー・エスケープ文節』
- 「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*」の『Unicode 関数 (CLI)』
- 「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*」の『DB2 CLI 用の SQLSTATES』
- 「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*」の『CLI アプリケーションでのパラメーター・マーカー・バインディング』
- 「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*」の『CLI アプリケーションのカーソル』

関連タスク:

- 「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*」の『CLI アプリケーションでのパラメーター・マーカーのバインディング』
- 「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*」の『CLI アプリケーションでのデータの更新と削除』

関連資料:

- 11 ページの『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』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』
- 「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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbmcon.c -- How to use multiple databases』
- 『dbuse.c -- How to use a database』

- 『tbmod.c -- How to modify table data』

SQLExecute 関数 (CLI) - ステートメントの実行

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLPrepare() は、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。

構文:

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

関数引き数:

表 47. SQLExecute 引き数

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

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、 DB2 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 SELECT ステートメント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取り出します。

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_EXECUTE 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 48. 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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカーソル』

関連タスク:

- 「*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*」の『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 「*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*」の『CLI アプリケーションでのパラメーター・マーカーのバインディング』
- 「*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*」の『CLI アプリケーションでのデータの更新と削除』

関連資料:

- 23 ページの『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』
- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 272 ページの『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 27 ページの『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』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『spclient.c -- Call various stored procedures』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』

SQLExtendedBind 関数 (CLI) - 列の配列のバインド

目的:

仕様:	DB2 CLI 6		

SQLExtendedBind() を使用して、 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。

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 ) ;
```

関数引き数:

表 49. SQLExtendedBind() 引き数

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

表 49. SQLExtendedBind() 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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。
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() の引き数 *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() の引き数 *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() の引き数 *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() の引き数 *DecimalDigits* と同じ目的を果たします。

使用法:

引き数 *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、 DB2 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、 DB2 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、据え置き引き数と見なされます。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、DB2 CLI が配列内の値を使用して実行する場合に、有効な値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 50. SQLExtendedBind() SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	*pfCType* 引き数の行によって識別されるデータ値から *pfParamType* 引き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。 (たとえば、SQL_C_DATE から SQL_DOUBLE への変換は無意味です。)
07009	無効な記述子索引	引き数 *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	引き数 *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 50. SQLExtendedBind() SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数の値が無効です。	引き数 *rgbValue* は NULL ポインターで、引き数 *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。	引き数 *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。	引き数 *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き数 *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_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 6 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。	DB2 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。

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 11 ページの『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』
- 27 ページの『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』
- 「*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*」の『CLI 関数戻りコード』

SQLExtendedFetch 関数 (CLI) - 拡張取り出し (行の配列の取り出し)

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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取り出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。

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 381 ページの『ステートメント属性 (CLI) のリスト』

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

目的:

仕様:	DB2 CLI 6.0		

SQLExtendedPrepare() は、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する場合に使用します。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 );
```

関数引き数:

表 51. SQLExtendedPrepare() 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*StatementText*	入力	SQL ステートメント・ストリング。
SQLINTEGER	*TextLength*	入力	*StatementText* 引き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、129 ページの『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 つの引き数は、SQLPrepare() の引き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文節を探さない。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文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 引き数は、*StatementText* 内のパラメーター・マーカーの数を指示します。

引き数 *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 つの引き数は、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

引き数 *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_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 52. 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 文節がありません。	*StatementText* の UPDATE ステートメントまたは DELETE ステートメントに、 WHERE 文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。	DB2 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、 DB2 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、DB2 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。	*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	引き数 *StatementText* の WHERE 文節に 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	引き数 *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
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 52. SQLExtendedPrepare 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、 *StatementText* に含まれている SQL ステートメントの実行がこの許可 ID に許可され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 文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可能になりました。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	引き数の値が無効です。	*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。

表 52. SQLExtendedPrepare 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。
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* 引き数が SQL_ATTR_IMP_ROW_DESC または SQL_ATTR_IMP_PARAM_DESC でした。 *Attribute* 引き数は 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。 (DB2 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。	引き数 *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。	引き数 *Attribute* に指定された値が、このバージョンの DB2 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。	引き数 *Attribute* に指定された値は、このバージョンの DB2 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。たとえば、 **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。

制限:

なし。

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』

- 「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」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 272 ページの『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「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」の『CLI 関数戻りコード』

SQLFetch 関数 (CLI) - 次の行の取り出し

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLFetch() は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。

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、取り出し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 */
```

関数引き数:

表 53. SQLFetch 引き数

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

使用法:

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可能) 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() 据え置き出力引き数 *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* 引き数と *cbValueMax* 引き数) 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。 DB2 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 NULL 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 NULL 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、 DB2 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() は次の行セットを取り出します。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() によって取り出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 は取り出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・バッファー内に取り出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、たとえば 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、その番号をそれぞれ 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。

引き数のマッピング

ドライバーは、引き数 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 54. 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

表 54. 記述子フィールド (続き)

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_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* 引き数
SQL_DESC_DATA_PTR	ARD	レコード	SQLBindCol() の *TargetValuePtr* 引き数
SQL_DESC_INDICATOR_PTR	ARD	レコード	SQLBindCol() の *StrLen_or_IndPtr* 引き数
SQL_DESC_OCTET_LENGTH	ARD	レコード	SQLBindCol() の *BufferLength* 引き数
SQL_DESC_OCTET_LENGTH_PTR	ARD	レコード	SQLBindCol() の *StrLen_or_IndPtr* 引き数
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* 引き数

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使用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* 引き数に渡されます。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取り出された場合、 SQL_NO_DATA_FOUND が返されます。

すべての行が取り出された場合、カーソルは結果表の末尾の後ろに置かれます。

診断:

表 55. SQLFetch SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられました。	1 つまたは複数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 つまたは複数の列の列番号が、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() の引き数 *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() の引き数 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 つまたは複数の列の数値を (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 Universal Database 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 DB2 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。
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。たとえば、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。

表 55. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説

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許可を持っていません。
		• 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、ディスクがいっぱいになりました。
		• 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。

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可能になりました。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 55. SQLFetch SQLSTATE (続き)

SQLSTATE	説明	解説
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。 *StatementHandle* で SQLPrepare() または SQLExecDirect() 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。
HY092	オプション・タイプが範囲外です。	直前の SQLBindFileToCol() 操作の *FileOptions* 引き数が無効でした。
HYC00	ドライバーが使用できません。	DB2 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。 DB2 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("    %-8d %-14.14s ¥n", deptnumb.val, location.val);

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

関連概念:

- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションにおける結果セットの用語』

関連タスク:

- 「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」の『CLI アプリケーションでの照会結果の取り出し』

関連資料:

- 11 ページの『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』
- 19 ページの『SQLBindFileToCol 関数 (CLI) - LOB 列への LOB ファイル参照のバインド』
- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 170 ページの『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 188 ページの『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 148 ページの『SQLFetchScroll() (CLI) のカーソル位置決めの規則』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『tbread.c -- How to read data from tables』

SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFetchScroll() は、結果セットからデータの指定された行セットを取り出し、すべてのバインドされた列にデータを返します。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。

構文:

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

関数引き数:

表 56. SQLFetchScroll 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLUSMALLINT	*FetchOrientation*	入力	取り出しのタイプ。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 詳細については、『カーソルの位置決め』を参照してください。
SQLINTEGER	*FetchOffset*	入力	取り出しを行う行の数。この引き数の解釈は、*FetchOrientation* 引き数の値によって異なります。詳細については、『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、これらの列に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、または取り出されているいくつかの行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* 引き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。たとえば前方向の静的カーソルは、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() によって取り出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 は取り出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。

バインドされた列にデータを返す

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

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、これらの機能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	行が正常に取り出されました。
SQL_ROW_SUCCESS_WITH_INFO	行が正常に取り出されました。しかし、行に関する警告が返されました。
SQL_ROW_ERROR	行を取り出し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取り出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、取り出した行の数を返すために使用します。これには、取り出し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、これらの関数は取り出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 57. 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、DB2 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 つまたは複数の列の列番号が、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、DB2 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 57. SQLFetchScroll 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。	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 データが取り出され、 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。
22003	数値が範囲外です。	1 つまたは複数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、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。
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、列にある値はそれぞれ無効である日付、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	取り出し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可能になりました。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 57. 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。	引き数 *FetchOrientation* に指定された値は無効でした。 引き数 *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、引き数 *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。	引き数 *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。	指定された取り出しタイプは、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);
```

関連概念:

- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションのカーソル』
- 「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」の『CLI アプリケーションにおける結果セットの用語』
- 「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」の『CLI アプリケーションでの診断の概説』

関連資料:

- 11 ページの『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』
- 127 ページの『SQLExtendedFetch 関数 (CLI) - 拡張取り出し (行の配列の取り出し)』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 188 ページの『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 243 ページの『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 322 ページの『SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 148 ページの『SQLFetchScroll() (CLI) のカーソル位置決めの規則』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『tbread.c -- How to read data from tables』

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* 引き数の値。

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 58. 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 59. 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
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 60. 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]

表 60. SQL_FETCH_RELATIVE 規則 (続き)

条件	新しい行セットの最初の行
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 61. 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*
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 62. 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 63. 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 64. 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

関連概念:

- 「*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*」の『CLI アプリケーションのカーソル』

関連資料:

- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』

SQLForeignKeys 関数 (CLI) - 外部キー列のリストの取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLForeignKeys() は、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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 65. SQLForeignKeys 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*PKCatalogName*	入力	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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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。
SQLCHAR *	*FKCatalogName*	入力	3 つの部分から成る外部基本表名のカタログ修飾子。ターゲットの DBMS が 3 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength4* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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。

表 65. SQLForeignKeys 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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() 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() 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() 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。これは、最大 1 つのキー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き数を指定しないと、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。

154 ページの『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。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) 分の余地を確保しておくか、または接続している DBMS がサポートしている TABLE_CAT、 TABLE_SCHEM、 TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別するために、 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() を呼び出すことができます。

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 以外の RDBMS に接続されると、以下の 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: DB2 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 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 66. 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。
HY009	引き数の値が無効です。	引き数 *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。

表 66. SQLForeignKeys 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。	名前の長さ引き数のうちの 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することができます。

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);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*管理ガイド: インプリメンテーション*」の『FOREIGN KEY 文節』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 277 ページの『SQLPrimaryKeys 関数 (CLI) - 表の主キー列の取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbconstr.c -- How to work with constraints associated with tables』

SQLFreeConnect 関数 (CLI) - 接続ハンドルの解放

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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);
```

関連資料:

- 100 ページの『SQLDisconnect 関数 (CLI) - データ・ソースからの切断』
- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』

SQLFreeEnv 関数 (CLI) - 環境ハンドルの解放

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』

関連資料:

• 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』

SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFreeHandle() は、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。

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 */
```

関数引き数:

表 67. SQLFreeHandle 引き数

データ・タイプ	引き数	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* が上記の値のいずれでもでない場合、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。DB2 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 68. 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* 引き数は SQL_HANDLE_DBC であり、 DB2 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。
HY010	関数のシーケンス・エラーです。	*HandleType* 引き数が 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* 引き数は 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* 引き数は 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* 引き数は 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* 引き数が 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* 引き数が、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");
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbmod.c -- How to modify table data』
- 『tbread.c -- How to read data from tables』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』

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)

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLFreeStmt() は、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。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 DB2 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 */
```

関数引き数:

表 69. SQLFreeStmt 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
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 DB2 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、SQL_HANDLE_STMT の *HandleType* セットを使用する SQLFreeHandle() 呼び出しと置き換えられました。このバージョンの DB2 CLI でも引き続きこのオプションをサポートしていますが、 SQLFreeHandle() の使用を DB2 CLI プログラムから開始するなら最新の規格に適合できるため、この方法をお勧めします。

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 70. 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。	引き数 *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;
}
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『ODBC アプリケーションでの LOB の使用法』
- 「*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*」の『CLI アプリケーションの記述子』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 59 ページの『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』
- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 251 ページの『SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得』
- 306 ページの『SQLSetCursorName 関数 (CLI) - カーソル名の設定』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『utilcli.c -- Utility functions used by DB2 CLI samples』

SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetConnectAttr() は、接続属性の現在の設定を返します。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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);
```

関数引き数:

表 71. *SQLGetConnectAttr* 引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
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。
SQLINTEGER	*BufferLength*	入力	• *ValuePtr* が文字ストリングを指す場合、この引き数の長さは **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* 引き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、 DB2 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。別の引き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 72. 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。	引き数 *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。	引き数 *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。	引き数 *Attribute* に指定された値は、このバージョンの DB2 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);
```

関連概念:

- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 366 ページの『接続属性 (CLI) リスト』
- 「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」の『CLI 関数戻りコード』

SQLGetConnectOption 関数 (CLI) - 接続オプションの現行設定値を戻す

使用すべきでない:

注:

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。

DB2 CLI のこのバージョンでは引き続き SQLGetConnectOption() をサポートしますが、 DB2 CLI プログラムでは、最新の標準に準拠して SQLGetConnectAttr() を使用するようお勧めします。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、Unicode 関数 (CLI) を参照してください。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 164 ページの『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』

- 「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」の『CLI 関数戻りコード』

SQLGetCursorName 関数 (CLI) - カーソル名の取得

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetCursorName() は、入力ステートメント・ハンドルに関連したカーソル名を返します。 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 73. SQLGetCursorName 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*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。
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、DB2 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 18 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 74. 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。引き数 *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。
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。	引き数 *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、 DB2 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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカーソル』

関連資料:

- 306 ページの『SQLSetCursorName 関数 (CLI) - カーソル名の設定』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『spserver.c -- Definition of various types of stored procedures』
- 『tbmod.c -- How to modify table data』

SQLGetData 関数 (CLI) - 列からのデータの取得

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

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。これは 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。

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 */
               SQLINTEGER        BufferLength,        /* cbValueMax */
               SQLINTEGER        *StrLen_or_IndPtr);  /* pcbValue */
```

関数引き数:

表 75. SQLGetData 引き数

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

表 75. SQLGetData 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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。
SQLINTEGER	*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。
SQLINTEGER *	*StrLen_or_IndPtr*	出力	DB2 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、DB2 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・ソースがこの機能をサポートします。
- 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・ソースがこの機能をサポートします。
- SQL_GD_BLOCK: SQL_GETDATA_EXTENSIONS *InfoType* 引き数の場合に 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 UDB (Unix および Windows 版) のデータ・ソースは、この機能をサポートします。

- 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 UDB は現在この機能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。

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、 DB2 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. 行を取り出します。
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 (一部の引き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 入力引き数は、 *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。
- DB2 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、 DB2 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。 DB2 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 NULL 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 (たとえば、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

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。

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 76. 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。	引き数 *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。引き数 *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。	引き数 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。	返された値は、引き数 *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。たとえば、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。
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 76. SQLGetData SQLSTATE (続き)

SQLSTATE	説明	解説
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。
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。	引き数 *BufferLength* の値は 0 より小さく、引き数 *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、DB2 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、 DB2 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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカーソル』

関連資料:

- 11 ページの『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』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』
- 『tbread.c -- How to read data from tables』
- 『tut_read.c -- How to read data from tables』

SQLGetDataLinkAttr 関数 (CLI) - DataLink 属性値の取得

目的:

仕様:	DB2 CLI 5.2		ISO CLI

データ・リンク値の現在の属性値を戻します。

構文:

```
SQLRETURN   SQLGetDataLinkAttr (
               SQLHSTMT          StatementHandle,    /* hStmt */
               SQLSMALLINT       Attribute,          /* fAttrType */
               SQLCHAR           *DataLink,          /* *pDataLink */
               SQLINTEGER        DataLinkLength,     /* cbDataLink */
               SQLPOINTER        *ValuePtr,          /* pAttribute */
               SQLINTEGER        BufferLength,       /* cbAttributeMax */
               SQLINTEGER        *StringLengthPtr);  /* *pcbAttribute */
```

関数引き数:

表 77. SQLGetDataLinkAttr 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	診断報告専用です。
SQLSMALLINT	*Attribute*	入力	抽出されるデータ・リンクの属性を識別します。有効値は以下のとおりです。 • SQL_ATTR_DATALINK_COMMENT • SQL_ATTR_DATALINK_LINKTYPE • SQL_ATTR_DATALINK_URLCOMPLETE (ファイルにアクセスするための完全な URL) • SQL_ATTR_DATALINK_URLPATH (ファイル・サーバー内のファイルにアクセスする場合) • SQL_ATTR_DATALINK_URLPATHONLY (ファイル・パスのみ) • SQL_ATTR_DATALINK_URLSCHEME • SQL_ATTR_DATALINK_URLSERVER
SQLCHAR *	*DataLink*	入力	抽出される属性が属する DATALINK 値。
SQLINTEGER	*DataLinkLength*	入力	*DataLink* 値の長さ。 *DataLink* 引き数内に NULL 終了ストリングが入っている場合、 *DataLinkLength* 用に SQL_NTS の値が渡されることがあります。
SQLPOINTER *	*ValuePtr*	出力	*Attribute* で指定される属性の値を返す先のメモリーを指すポインター。
SQLINTEGER	*BufferLength*	入力	*ValuePtr* が戻り値を保存するのに使用できるストレージ量。
SQLINTEGER *	*StringLength*	出力	**Attribute* 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。 *Attribute* が NULL ポインターである場合、長さは戻されません。戻りに使用できるバイト数が、NULL 終止符文字の長さを引いた BufferLength より大きい場合は、 SQLSTATE HY090 が戻されます。

使用法:

この関数は、データベースから検索された DATALINK 値、または SQLBuildDataLink() を使用して構築された DATALINK 値と一緒に使用されます。 *AttrType* 値は、戻される DATALINK 値から属性を判別します。 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。

戻りコード:

- SQL_SUCCESS
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 78. SQLGetDataLinkAttr 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。)
HY000	一般的なエラーです。	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。
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。)
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	引き数の値が無効です。	引き数 **DataLink* に指定された値は、NULL ポインター、または無効でした。
HY090	ストリングまたはバッファーの長さが無効です。	引き数 *BufferLength* に指定された値は 0 より小さい値でした。または引き数 *DataLinkLength* に指定された値は 0 より小さく、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。	引き数 *AttrType* に指定された値は、無効でした。

制限:

なし。

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 48 ページの『SQLBuildDataLink 関数 (CLI) - DATALINK 値の作成』
- 「*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*」の『CLI 関数戻りコード』

SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetDescField() は、記述子レコードの単一フィールドの現行設定値を返します。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 79. SQLGetDescField 引き数

データ・タイプ	引き数	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探索す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* 引き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、この引き数の長さは **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* 引き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 DB2 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可能です。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 80. 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。)
07009	記述子索引が無効です。	*RecNumber* 引き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 DB2 CLI 定義済みフィールドではありませんでした。 *FieldIdentifier* 引き数はレコード・フィールドであり、 *RecNumber* 引き数は 0 でした。 *RecNumber* 引き数は 0 より小さく、フィールドはヘッダー・フィールドまたは DB2 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、DB2 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。

表 80. SQLGetDescField 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。	名前の長さ引き数のうちの 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* 引き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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションの記述子』
- 「*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*」の『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 184 ページの『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』

- 243 ページの『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 414 ページの『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 401 ページの『記述子 FieldIdentifier 引き数の値 (CLI)』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』

SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetDescRec() は、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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
               SQLINTEGER        *LengthPtr,         /* *Length */
               SQLSMALLINT       *PrecisionPtr,      /* *Precision */
               SQLSMALLINT       *ScalePtr,          /* *Scale */
               SQLSMALLINT       *NullablePtr);      /* *Nullable */
```

関数引き数:

表 81. SQLGetDescRec 引き数

データ・タイプ	引き数	使用法	説明
SQLHDESC	*DescriptorHandle*	入力	記述子ハンドル。

表 81. SQLGetDescRec 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLSMALLINT	*RecNumber*	入力	アプリケーションが探索す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* 引き数は、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、 DB2 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。
SQLINTEGER *	*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き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。たとえば、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 82. 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。)

表 82. SQLGetDescRec SQLSTATE (続き)

SQLSTATE	説明	解説
07009	記述子索引が無効です。	*RecNumber* 引き数は 0 に設定され、 *DescriptorHandle* 引き数は IPD ハンドルでした。 *RecNumber* 引き数は 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* 引き数は 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、DB2 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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションの記述子』

関連資料:

- 11 ページの『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』
- 179 ページの『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 315 ページの『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』
- 27 ページの『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』
- 414 ページの『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』

SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetDiagField() は、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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 83. SQLGetDiagField 引き数

データ・タイプ	引き数	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	*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	アプリケーションが探索す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* 引き数が診断ヘッダー・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	値を戻したい診断データ構造のフィールドを示します。詳細については、192 ページのDiagIdentifier 引き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 83. SQLGetDiagField 引き数 (続き)

データ・タイプ	引き数	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 DB2 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 SQLCHAR エレメントの数 (またはこの関数の Unicode 版の場合は SQLWCHAR エレメントの数) 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、この引き数は無視されます。

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)。

DB2 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* 引き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 引き数

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* 引き数は、*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の下に示されます。

DB2 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 引き数

この引き数は、診断データ構造にある希望するフィールドの 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* 引き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、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**。 DB2 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。切り捨てが発生したかどうかを判別するには、アプリケーションは *BufferLength* を、 **StringLengthPtr* に書き込まれた使用可能な実際のバイト数と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: 以下のどちらかです。
  - *DiagIdentifier* 引き数は、有効な値の 1 つではありませんでした。
  - *DiagIdentifier* 引き数は、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* 引き数が負か 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。

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでの診断の概説』

関連資料:

- 194 ページの『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 423 ページの『DiagIdentifier 引き数 (CLI) のヘッダー・フィールドとレコード・フィールド』
- 「*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*」の『CLI 関数戻りコード』

SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetDiagRec() は、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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 84. SQLGetDiagRec の引き数

データ・タイプ	引き数	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	*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	アプリケーションが探索す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。

表 84. SQLGetDiagRec の引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*SQLState*	出力	診断レコード *RecNumber* に関する 5 文字の SQLSTATE コード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。
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:

DB2 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* 引き数が 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* 引き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、使用可能な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 引き数

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* 引き数は、*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* 引き数のヘッダー・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* に書き込まれる使用可能な実際の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: 以下のどちらかです。
  - *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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 188 ページの『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 423 ページの『DiagIdentifier 引き数 (CLI) のヘッダー・フィールドとレコード・フィールド』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『spclient.c -- Call various stored procedures』
- 『utilcli.c -- Utility functions used by DB2 CLI samples』

SQLGetEnvAttr 関数 (CLI) - 現行の環境属性値の検索

目的:

仕様:	DB2 CLI 2.1		ISO CLI

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

これらのオプションは、SQLSetEnvAttr() 関数を使用して設定されます。

構文:

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

関数引き数:

表 85. SQLGetEnvAttr 引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*EnvironmentHandle*	入力	環境ハンドル。
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、 DB2 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、DB2 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 86. 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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのハンドルの解放』

関連タスク:

- 「*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*」の『CLI アプリケーションの初期設定』

関連資料:

- 319 ページの『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 361 ページの『環境属性 (CLI) リスト』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『cli_info.c -- How to get and set environment attributes at the client level』

SQLGetFunctions 関数 (CLI) - 関数の取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

SQLGetFunctions() を使って、個々の DB2 CLI または ODBC 関数がサポートされているかどうかを照会することができます。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。

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 */
```

関数引き数:

表 87. SQLGetFunctions 引き数

データ・タイプ	引き数	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 87. SQLGetFunctions 引き数 (続き)

データ・タイプ	引き数	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 88. 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。

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbinfo.c -- How to get and set information at the database level』
- 『dbinfo.out -- HOW TO GET AND SET DATABASE INFORMATION (CLI)』
- 『ilinfo.c -- How to get information at the installation image level』
- 『ininfo.c -- How to get information at the instance level』

SQLGetInfo 関数 (CLI) - 一般情報の取得

目的:

仕様:	DB2 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 DBMS に関する一般情報を戻します。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 89. SQLGetInfo 引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	データベース接続ハンドル

表 89. SQLGetInfo 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUSMALLINT	*InfoType*	入力	任意の情報タイプ。この引き数に指定できる値は、203 ページの『SQLGetInfo() から戻される情報』に説明されています。
SQLPOINTER	*InfoValuePtr*	出力 (入力兼用)	この関数が所定の情報を保管す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* 引き数が SQL_DRIVER_HDESC または SQL_DRIVER_HSTMT である場合、 *InfoValuePtr* 引き数は入力にも出力にもなります。
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* 引き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、このサイズには NULL 終止符文字が含まれません。 *StringLengthPtr* が指すロケーションの値が *BufferLength* に指定されている *InfoValuePtr* バッファーのサイズより大きい場合、ストリング出力情報が *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 の有効値のリストと、その値に関して `SQLGetInfo()` が戻す情報の詳細は、 203 ページの『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 90. 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* 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。引き数 *StringLengthPtr* 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 を返します。)
08003	接続がクローズされています。	*InfoType* で要求されているタイプの情報には、オープン接続が必要です。オープン接続が必要ないのは、SQL_ODBC_VER だけです。
40003 **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。	引き数 *BufferLength* に指定された値は、0 より小さい値でした。
HY096	情報タイプが範囲外です。	無効な *InfoType* が指定されました。
HYC00	ドライバーが使用できません。	引き数 *InfoType* に指定された値は、 DB2 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: DB2 CLI は、この表の *InfoType* ごとに値を返します。 *InfoType* 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 特権をユーザーが確保することを示します。 ″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)
総計機能のビット・マスク列挙サポートは以下のとおりです。
- 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)
DB2 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、DB2 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文節を 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。

  DB2 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。 DB2 CLI は常に SQL_BRC_ROLLED_UP を返し、連続した INSERT、DELETE、または UPDATE ステートメントの行カウントが 1 つにロールアップされることを示します。

ODBC はさらに、DB2 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もブックマークは有効です。
- SQL_BP_DELETE。行のブックマークは、その行が削除された後に有効です。

- 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。ブックマークは、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に有効です。
- SQL_BP_UPDATE。行のブックマークは、その行のいずれかの列 (キー列を含む) が更新された後に有効です。
- 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。 DB2 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 はさらに値 SQL_CL_END も定義しますが、DB2 CLI はこの値を返しません。

DB2 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。

DB2 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)
修飾子 (カタログ) 用のデータベース・ベンダーの用語。

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。

DB2 CLI では 3 つの部分の名前がサポートされていないので、長さゼロのストリングが返されます。

DB2 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)
これは、カタログに使われることを除き、SQL_SCHEMA_USAGE に似ています。

カタログの使用が可能であって、しかも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_CU_DML_STATEMENTS - カタログはすべての DML ステートメントでサポートされています。
- SQL_CU_INDEX_DEFINITION - カタログはすべての索引定義ステートメントでサポートされています。
- SQL_CU_PRIVILEGE_DEFINITION - カタログはすべての特権定義ステートメントでサポートされています。

- SQL_CU_PROCEDURE_INVOCATION - カタログは ODBC 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_CU_TABLE_DEFINITION - カタログはすべての表定義ステートメントでサポートされています。

データ・ソースでカタログがサポートされていない場合、ゼロの値が戻されます。

DB2 CLI の前のバージョンでは、この *InfoType* は SQL_QUALIFIER_USAGE でした。

SQL_COLLATION_SEQ (ストリング)
照合シーケンスの名前。これは、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名 (たとえば ISO 8859-1 または EBCDIC) を示す文字ストリングです。これが不明であると、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 - 結果が NULL 値であることを示します (IBM RDBMS の場合)。
- SQL_CB_NON_NULL - 結果が非 NULL 列値が連結されたものであることを示します。

SQL_CONVERT_BIGINT
SQL_CONVERT_BINARY
SQL_CONVERT_BIT
SQL_CONVERT_CHAR
SQL_CONVERT_DATE
SQL_CONVERT_DECIMAL
SQL_CONVERT_DOUBLE
SQL_CONVERT_FLOAT
SQL_CONVERT_INTEGER
SQL_CONVERT_INTERVAL_YEAR_MONTH
SQL_CONVERT_INTERVAL_DAY_TIME
SQL_CONVERT_LONGVARBINARY
SQL_CONVERT_LONGVARCHAR
SQL_CONVERT_NUMERIC
SQL_CONVERT_REAL
SQL_CONVERT_SMALLINT
SQL_CONVERT_TIME
SQL_CONVERT_TIMESTAMP
SQL_CONVERT_TINYINT
SQL_CONVERT_VARBINARY
SQL_CONVERT_VARCHAR

SQL_CONVERT_WCHAR
SQL_CONVERT_WLONGVARCHAR
SQL_CONVERT_WVARCHAR

(上記はすべて 32 ビット・マスク)

CONVERT スカラー関数を用いて行われる、 *InfoType* に名前をあげら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タイプのデータ変換をサポートしません。

たとえば、データ・ソースが SQL_INTEGER データから SQL_DECIMAL データ・タイプへの変換をサポート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* とともに `SQLGetInfo()` を呼び出します。アプリケーションは次に、返されたビット・マスクを SQL_CVT_DECIMAL で AND 演算します。結果値が非ゼロであれば、変換はサポートされています。

次のビット・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 ビット無符号整数)

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。

DB2 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。相関名はサポートされていて、任意の有効なユーザー定義名にすることができます。
- 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文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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_SCHEMA (32 ビット・マスク)

CREATE SCHEMA ステートメント中のどの文節を 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文節を 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、どの文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文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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文節を 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。カーソルは破棄され、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。カーソルは破棄されますが、動的 SQL ステートメントに関するアクセス・プランは保存されます (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・プランは保存されます (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取り出すか、またはカーソルをクローズしてステートメントを再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。カーソルは破棄され、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。カーソルは破棄されますが、動的 SQL ステートメントに関するアクセス・プランは保存されます (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・プランは保存されます (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取り出すか、またはカーソルをクローズしてステートメントを再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 ビット無符号整数)

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・ソース名の付いた文字ストリング。アプリケーションが SQLConnect() を呼び出す場合、これが *szDSN* 引き数の値になります。アプリケーションが SQLDriverConnect() または SQLBrowseConnect() を呼び出した場合、これ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 キーワードがなかった場合は、これ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 ビット無符号整数)
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 OS/390 や DB2 for OS/400 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、DB2 Connect ゲートウェイでの CATALOG DCS DATABASE DIRECTORY コマンドの発行時に指定された DCS データベース名になります。

SQL_DATETIME_LITERALS (32 ビット無符号整数)
DBMS がサポートする、日時リテラルを示します。 DB2 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、DB2 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。

たとえば、次のようになります。
- ″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)。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 ビット無符号整数)
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)。

  これは IBM 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)。

  これは IBM のカーソル安定度レベルと同等です。
- SQL_TXN_REPEATABLE_READ = トランザクションは、探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)。

  これは IBM 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 (非コミット読み取りはできますが、幻像読み取りはできません)。

  これは IBM 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 DB2 UDB for AS/400 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_HDBC (32 ビット)
DB2 CLI のデータベース・ハンドル

SQL_DRIVER_HDESC (32 ビット)
DB2 CLI の記述子ハンドル

SQL_DRIVER_HENV (32 ビット)
DB2 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)
DB2 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 (つまり、 `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* は入力引き数と出力引き数の両方です。 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 DB2 CLI 特定関数 (LOB 関数など) を呼び出したい場合、そのアプリケーションは、DB2 CLI ライブラリーをロードし、オペレーティング・システム呼び出しを発行して希望の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)
DB2 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)
ドライバーがサポートする ODBC のバージョン番号。 DB2 CLI は ″03.00″ を返します。

SQL_DRIVER_VER (ストリング)
CLI ドライバーのバージョン。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)。たとえば、 ″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。

SQL_DROP_ASSERTION (32 ビット無符号整数)
DROP ASSERTION ステートメント中のどの文節を DBMS がサポートしているかを示します。 DB2 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、DB2 CLI が返さない以下の値を定義します。
- SQL_DA_DROP_ASSERTION

SQL_DROP_CHARACTER_SET (32 ビット無符号整数)
DROP CHARACTER SET ステートメント中のどの文節を DBMS がサポー

トしているかを示します。 DB2 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、DB2 CLI が返さない以下の値を定義します。
- SQL_DCS_DROP_CHARACTER_SET

SQL_DROP_COLLATION (32 ビット無符号整数)
DROP COLLATION ステートメント中のどの文節を DBMS がサポートしているかを示します。 DB2 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、DB2 CLI が返さない以下の値を定義します。
- SQL_DC_DROP_COLLATION

SQL_DROP_DOMAIN (32 ビット無符号整数)
DROP DOMAIN ステートメント中のどの文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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_SCHEMA (32 ビット無符号整数)
DROP SCHEMA ステートメント中のどの文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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 ビット無符号整数)
DROP TABLE ステートメント中のどの文節を DBMS がサポートしているかを示します。
- SQL_DT_DROP_TABLE
- SQL_DT_CASCADE
- SQL_DT_RESTRICT

SQL_DROP_TRANSLATION (32 ビット無符号整数)
DROP TRANSLATION ステートメント中のどの文節を DBMS がサポートしているかを示します。 DB2 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、DB2 CLI が返さない以下の値を定義します。
- SQL_DTR_DROP_TRANSLATION

SQL_DROP_VIEW (32 ビット無符号整数)
DROP VIEW ステートメント中のどの文節を DBMS がサポートしているかを示します。 DB2 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、DB2 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 ビット無符号マスク)
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。 DB2 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)
DB2 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)。
- 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)
DB2 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)。
- 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取り出し方向。

以下のビット・マスクとフラグを使って、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。 DB2 CLI ドライバーは単一階層の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、DB2 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)
DB2 CLI がサポートする転送専用カーソルの属性を示します (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)
DB2 CLI がサポートする転送専用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_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。DB2 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、DB2 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)

サーバーでサポートされる GROUP BY 文節の程度を示します。

- SQL_GB_NO_RELATION。 GROUP BY の列と SELECT リストの列の間の関係はありません。
- SQL_GB_NOT_SUPPORTED。 GROUP BY はサポートされていません。
- SQL_GB_GROUP_BY_EQUALS_SELECT。 GROUP BY には、選択リスト中のすべての非集約列が組み込まれていなければなりません。
- SQL_GB_GROUP_BY_CONTAINS_SELECT。 GROUP BY 文節には、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 文節を列の各グループ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 (たとえば、表名) の大文字小文字の区別を示します。

値は、以下のとおりです。

- SQL_IC_UPPER = ID 名は大文字でシステム・カタログに保管されます。
- SQL_IC_LOWER = ID 名は小文字でシステム・カタログに保管されます。
- SQL_IC_SENSITIVE = ID 名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 = ID 名は大文字小文字の区別がなく、混合文字でシステム・カタログに保管されます。

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)
サポートされる、CREATE INDEX ステートメント中のキーワードを示します。

- 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()` を、 SQL_DLL_INDEX *InfoType* 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。 DB2 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、DB2 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。

- 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・ソースが保全性拡張機能 (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。

DB2 CLI の前のバージョンでは、この *InfoType* は SQL_ODBC_SQL_OPT_IEF でした。

SQL_KEYSET_CURSOR_ATTRIBUTES1 (32 ビット・マスク)
DB2 CLI がサポートするキー・セット・カーソルの属性を示します (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)
DB2 CLI がサポートするキー・セット・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_KEYWORDS (ストリング)
データ・ソース固有のすべてのキーワードをコンマで区切ったリストを収めた文字ストリング。これは、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・ソースがサポートし、しかも LIKE 述部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場合は ″Y″。それ以外の場合は ″N″。

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 ビット無符号整数)
DB2 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 ビット無符号整数)
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。たとえばバイナリー・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。

DB2 CLI の前のバージョンでは、この *fInfoType* は SQL_MAX_QUALIFIER_NAME_LEN でした。

SQL_MAX_CHAR_LITERAL_LEN (32 ビット無符号整数)
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 文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 文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)
サーバーが選択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)
DB2 CLI ドライバー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。

DB2 CLI の前のバージョンでは、この *InfoType* 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。

ゼロが返された場合は、限度はシステム・リソースによって異なることを示しています。

DB2 CLI の前のバージョンでは、この *InfoType* 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 ビット無符号整数)
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_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 ビット無符号整数)
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* 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)。

DB2 CLI の前のバージョンでは、この *fInfoType* は SQL_MAX_OWNER_NAME_LEN でした。

SQL_MAX_STATEMENT_LEN (32 ビット無符号整数)
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 文節に使用できる表名の最大数を示します。

SQL_MAX_USER_NAME_LEN (16 ビット整数)
<user identifier> で使用できる最大サイズを示します (バイト単位)。

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)
文字ストリング ″Y″ は、複数の接続に関するアクティブ・トランザクションを持つことができることを示し、 ″N″ は、一度に 1 つの接続だけがアクティブ・トランザクションを持つことができることを示します。

DB2 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。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。
- SQL_NNC_NON_NULL。列を NOT NULL として定義できます。
- SQL_NNC_NULL。列を NOT NULL として定義できません。

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。
- SQL_NC_HIGH。 NULL 値は高位にソートされます。
- SQL_NC_LOW。 NULL 値が低位にソートされることを示します。

SQL_NUMERIC_FUNCTIONS (32 ビット・マスク)
サポートされている ODBC スカラー数字関数を示します。この関数は、ODBC ベンダー・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。

次のビット・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。

- SQL_OAC_NONE
- SQL_OAC_LEVEL1
- SQL_OAC_LEVEL2

SQL_ODBC_INTERFACE_CONFORMANCE (32 ビット無符号整数)

DB2 CLI ドライバーが準拠している 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。上記のようなごく基本的な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機能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。

値は、以下のとおりです。

- 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)

値は、以下のとおりです。

- 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。

DB2 CLI はストリング ″03.01.0000″ 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 文節の列の基礎となる表の順序は、JOIN 文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 文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 文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、 DB2 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 ビット無符号整数)
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。これは概念的には、SQL ステートメントのバッチを生成するドライバーと同等であり、配列内の各パラメーター・セットごとに 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。
- SQL_PARC_NO_BATCH。使用可能な行カウントは 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。これは概念的には、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 ビット無符号整数)
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。
- SQL_PAS_BATCH。パラメーターのセットにつき使用可能な結果セットは 1 つです。これは概念的には、SQL ステートメントのバッチを生成するドライバーと同等であり、配列内の各パラメーター・セットごとに 1 つあります。

- SQL_PAS_NO_BATCH。使用可能な結果セットは 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。これは概念的には、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、ドライバーが許可しません。

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、カーソルを介して列を更新できるようにするために、サーバーが FOR UPDATE 文節を <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 DB2 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)
以下の値を返します。
- SQL_IC_UPPER - SQL 中の引用符付き ID は大文字小文字の区別がなく、大文字でシステム・カタログに保管されます。
- SQL_IC_LOWER - SQL 中の引用符付き ID は大文字小文字の区別がなく、小文字でシステム・カタログに保管されます。
- 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 - SQL 中の引用符付き ID は大文字小文字の区別がなく、混合文字でシステム・カタログに保管されます。

これを、(引用符なし) ID がシステム・カタログに格納される方法を判別するのに使用される、 SQL_IDENTIFIER_CASE *InfoType* 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・ドライバーまたは混合カーソルは、取り出されたすべての行のバージョンまたは値を保守するので、行の最後の取り出し以後に行に対してユーザーによって加えられた更新しか検出できないことを意味します。 (これに該当するのは、削除や追加ではなく、更新だけです。) SQLFetchScroll() を呼び出した場合、ドライバーは SQL_ROW_UPDATED フラグを行状況配列に戻すことがあります。上記の場合以外、″N″ です。

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。

DB2 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)
SQL ステートメントの実行時に、関連したスキーマ (所有者) のあるステートメントのタイプを示します。スキーマ修飾子 (所有者) は以下のとおりです。

- SQL_SU_DML_STATEMENTS - すべての DML ステートメントでサポートされています。
- SQL_SU_PROCEDURE_INVOCATION - プロシージャー呼び出しステートメントでサポートされています。
- SQL_SU_TABLE_DEFINITION - すべての表定義ステートメントでサポートされています。
- SQL_SU_INDEX_DEFINITION - すべての索引定義ステートメントでサポートされています。
- SQL_SU_PRIVILEGE_DEFINITION - すべての特権定義ステートメント (つまり、 GRANT ステートメントと取り消しステートメント) でサポートされています。

DB2 CLI の前のバージョンでは、この *InfoType* 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って、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_READ_ONLY
- SQL_SCCO_LOCK
- SQL_SCCO_TIMESTAMP
- SQL_SCCO_VALUES

DB2 CLI は、SQL_SCCO_LOCK を戻します。これは、ロックのレベルについて、行を確実に更新できるもののうち最低レベルのものが使用されていることを示します。

SQL_SCROLL_OPTIONS (32 ビット・マスク)
両方向スクロール・カーソル用にサポートされているスクロール・オプション。

以下のビット・マスクとフラグを使って、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: ドライバーは、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: ドライバーは、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: ドライバーは、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() や SQLColumns() 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のに使用します。

SQL_SERVER_NAME (ストリング)
DB2 インスタンスの名前。これは SQL_DATA_SOURCE_NAME (データベース・サーバーの実際の名前) とは対照的です。 (DBMS の中には、実際のデータベースのサーバー名と違う名前を CONNECT 接続に指定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、または索引名などの ID 名内で使用できるすべての特殊文字 (つまり、 a...z、A...Z、0...9、および下線以外のすべての文字) を使用した文字ストリング。たとえば ″@#″ などがあります。 ID 中でこのような文字を 1 つ以上使用する場合、その ID は区切り ID でなければなりません。

SQL_SQL_CONFORMANCE (32 ビット無符号整数)
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)
DB2 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。
- 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。
- 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。
- 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文節 (SQL-92 により定義) を示します。

- 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)

DB2 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。

- 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。

- 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。

- 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文節 (SQL-92 により定義) を示します。

- 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。

- SQL_SRVC_VALUE_EXPRESSION
- SQL_SRVC_NULL
- SQL_SRVC_DEFAULT
- SQL_SRVC_ROW_SUBQUERY

SQL_SQL92_STRING_FUNCTIONS (32 ビット・マスク)
DB2 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。

- SQL_SSF_CONVERT
- SQL_SSF_LOWER
- SQL_SSF_UPPER
- SQL_SSF_SUBSTRING
- SQL_SSF_TRANSLATE
- SQL_SSF_TRIM_BOTH
- SQL_SSF_TRIM_LEADING
- SQL_SSF_TRIM_TRAILING

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。

- SQL_SVE_CASE
- SQL_SVE_CAST
- SQL_SVE_COALESCE
- SQL_SVE_NULLIF

SQL_STANDARD_CLI_CONFORMANCE (32 ビット・マスク)

DB2 CLI が準拠している 1 つまたは複数の CLI 標準を示します。

- SQL_SCC_XOPEN_CLI_VERSION1
- SQL_SCC_ISO92_CLI

SQL_STATIC_CURSOR_ATTRIBUTES1 (32 ビット・マスク)

DB2 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 1/2)。

- SQL_CA1_NEXT
- SQL_CA1_ABSOLUTE
- SQL_CA1_RELATIVE
- SQL_CA1_BOOKMARK
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_EXCLUSIVE
- 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_STATIC_CURSOR_ATTRIBUTES2 (32 ビット・マスク)

DB2 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。

- SQL_SS_ADDITIONS: カーソルは追加された行を認識できます。カーソルはこれらの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。カーソルは、削除された行からスクロールした後にその行に戻れません。
- SQL_SS_UPDATES: カーソルは追加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。

次のビット・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* 引き数を指定した 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* 引き数を指定した 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。

- 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、相互に関連しあった副照会をサポートします。
- SQL_SQ_EXISTS - *exist* (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)
データベース・ベンダーの表に対する用語

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 か DML、またはその両方が含まれているかどうかを示します。

- SQL_TC_NONE = トランザクションはサポートされていません。
- SQL_TC_DML = トランザクションは DML ステートメント (SELECT、INSERT、UPDATE、DELETE など) だけを含むことができます。トラン

ザクション中で DDL ステートメント (CREATE TABLE、DROP INDEX、その他) が検出されるとエラーになります。
- 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。
- SQL_U_UNION - UNION 文節をサポートしています。
- SQL_U_UNION_ALL - UNION 文節の 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。これは `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。

関連概念:

- 「*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*」の『CLI アプリケーションでのベンダー・エスケープ文節』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカーソル』

関連資料:

- 251 ページの『SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得』
- 「*SQL リファレンス 第 2 巻*」の『CALL ステートメント』
- 「*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*」の『CLI 関数戻りコード』
- 「*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*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

関連サンプル:

- 『ilinfo.c -- How to get information at the installation image level』
- 『ininfo.c -- How to get information at the instance level』

SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し

目的:

仕様:	DB2 CLI 2.1		

SQLGetLength() 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。

構文:

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

関数引き数:

表 91. SQLGetLength 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。これは、すでに割り振り済みのものか、または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	LOB ロケーター値に設定する必要があります。

表 91. SQLGetLength 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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 引き数には、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。

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 92. 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。

表 92. SQLGetLength 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。	*LocatorCType* は、SQL_C_CLOB_LOCATOR、SQL_C_BLOB_LOCATOR、または SQL_C_DBCLOB_LOCATOR のいずれでもありません。
HY009	引き数の値が無効です。	*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* 出力引き数を調べて、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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのラージ・オブジェクトの使用』

- 「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」の『CLI アプリケーションでの LOB ロケーター』

関連資料:

- 247 ページの『SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』
- 『spserver.c -- Definition of various types of stored procedures』

SQLGetPosition 関数 (CLI) - ストリングの開始位置を戻す

目的:

仕様:	DB2 CLI 2.1		

SQLGetPosition() は、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、パターン値は 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。

ソースおよび探索 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。

構文:

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

関数引き数:

表 93. SQLGetPosition 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。これは、すでに割り振り済みのものか、または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

表 93. SQLGetPosition 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、この引き数は無視されます。
SQLCHAR *	*SearchLiteral*	入力	この引き数は、探索ストリング・リテラルを収容するストレージを指します。 *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* 内のストリングの長さ (バイト単位)。[a] この引き数値が 0 の場合は、引き数 *SearchLocator* は有効です。
SQLUINTEGER	*FromPosition*	入力	BLOB や CLOB の場合、これはソース・ストリング内で探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探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) 引き数には、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 94. 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。
42818	演算子または関数のオペランドに互換性がありません。	パターンの長さは、以下のような変数 SQL データ・タイプに関連付けられた最大データ長より長くなります。 • SQL_C_CLOB_LOCATOR の LocatorCType の場合のリテラルの最大サイズは、SQLVARCHAR の最大サイズになります。 • SQL_C_BLOB_LOCATOR の LocatorCType の場合のリテラルの最大サイズは、SQLVARBINARY の最大サイズになります。 • SQL_C_DBLOB_LOCATOR の LocatorCType の場合のリテラルの最大サイズは、SQLVARGRAPHIC の最大サイズになります。
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。
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	引き数の値が無効です。	*LocatedAt* 引き数を指すポインターが NULL でした。 *FromPosition* の引き数値は、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。

表 94. SQLGetPosition 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。	*SearchLiteralLength* の値が 1 より小さく、かつ SQL_NTS ではありませんでした。
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* 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* 出力引き数を調べて、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 starting position of the CLOB piece of data */
cliRC = SQLGetPosition(hstmtLocUse,
                       SQL_C_CLOB_LOCATOR,
                       clobLoc,
                       0,
                       (SQLCHAR *)"Interests",
                       strlen("Interests"),
                       1,
                       &clobPiecePos,
                       &ind);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのラージ・オブジェクトの使用』
- 「*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*」の『CLI アプリケーションでの LOB ロケーター』

関連資料:

- 236 ページの『SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し』
- 247 ページの『SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し』
- 「*SQL リファレンス 第 2 巻*」の『FREE LOCATOR ステートメント』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』
- 『spserver.c -- Definition of various types of stored procedures』

SQLGetSQLCA 関数 (CLI) - SQLCA データ構造の取得

使用すべきでない:

注:

SQLGetSQLCA() は使用すべきでない関数になりました。

このバージョンの DB2 CLI では引き続き SQLGetSQLCA() をサポートしますが、 DB2 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。

SQLGetDiagField() および SQLGetDiagRec() を使用して、診断情報を探してください。

関連概念:

- 「*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*」の『CLI アプリケーションでの診断の概説』

関連資料:

- 188 ページの『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』
- 194 ページの『SQLGetDiagRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』

関連サンプル:

- 『clisqlca.c -- How to retrieve SQLCA-equivalent information 』

SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetStmtAttr() は、ステートメント属性の現行設定値を戻します。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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);
```

関数引き数:

表 95. SQLGetStmtAttr 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLINTEGER	*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、この引き数の長さは **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 DB2 CLI 属性の場合、アプリケーションは *BufferLength* 引き数を設定して、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 SQLCHAR エレメントの数 (またはこの関数の Unicode 版の場合は SQLWCHAR エレメントの数) 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* 引き数は偶数でなければなりません。
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、 DB2 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 引き数にそのストリングの最大長を指定し、 DB2 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 引き数と StringLengthPtr 引き数は使用されません。

次の引き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 96. 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。	引き数 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。
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。

表 96. SQLGetStmtAttr 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。	引き数 *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。	引き数 *Attribute* に指定された値が、このバージョンの DB2 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* 引き数は 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。	引き数 *Attribute* で指定された値はこのバージョンの DB2 CLI の有効な DB2 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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 164 ページの『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbinfo.c -- How to get and set information at the database level』
- 『dbuse.c -- How to use a database』

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

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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);
```

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』
- 243 ページの『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』

SQLGetSubString 関数 (CLI) - ストリング値の部分的な取り出し

目的:

仕様:	DB2 CLI 2.1		

SQLGetSubString() は、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すのに使用されます。

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);
```

関数引き数:

表 97. SQLGetSubString 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。これは、すでに割り振り済みのものか、または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。
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 の場合は単一バイト・スペース、 DBCLOB の場合は 2 バイト・スペース)。
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。

表 97. SQLGetSubString 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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取り出します。

Locator 引き数には、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 98. 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。

表 98. SQLGetSubString SQLSTATE (続き)

SQLSTATE	説明	解説
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 (たとえば、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	引き数の値が無効です。	*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。
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* 出力引き数を調べて、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:

```
/* 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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『ODBC アプリケーションでの LOB の使用法』

関連資料:

- 11 ページの『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』
- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 170 ページの『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 236 ページの『SQLGetLength 関数 (CLI) - ストリング値の長さの取り出し』
- 239 ページの『SQLGetPosition 関数 (CLI) - ストリングの開始位置を戻す』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『spserver.c -- Definition of various types of stored procedures』

SQLGetTypeInfo 関数 (CLI) - データ・タイプ情報の取得

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetTypeInfo() は、 DB2 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。

構文:

```
SQLRETURN   SQLGetTypeInfo (
               SQLHSTMT         StatementHandle,  /* hstmt */
               SQLSMALLINT      DataType);        /* fSqlType */
```

関数引き数:

表 99. SQLGetTypeInfo 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLSMALLINT	*DataType*	入力	照会される 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 • 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 • SQL_TIMESTAMP • SQL_TINYINT • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC 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。

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* 引き数には 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、次に説明します。

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。たとえば ″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 (たとえば、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)。

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 (たとえば、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き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 文節でのデータ・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 文節で使用できない場合。
- SQL_LIKE_ONLY : WHERE 文節で、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 文節で、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 文節で、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 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 (DB2 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 (DB2 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 (DB2 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 100. 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。

表 100. SQLGetTypeInfo 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。
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);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカーソル』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 11 ページの『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』
- 141 ページの『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 201 ページの『SQLGetInfo 関数 (CLI) - 一般情報の取得』
- 298 ページの『SQLSetColAttributes 関数 (CLI) - 列属性の設定』
- 「*SQL リファレンス 第 2 巻*」の『ALTER TABLE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CREATE TABLE ステートメント』
- 「*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*」の『CLI 関数戻りコード』
- 「*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*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

関連サンプル:

- 『dtinfo.c -- How get information about data types』

SQLMoreResults 関数 (CLI) - さらに結果セットがあるかどうかの判別

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLMoreResults() は、以下のものに関連付けられているステートメント・ハンドルでさらに利用可能な情報があるかどうかを判別します。

- 照会のパラメーター値の配列入力
- 結果セットを戻しているストアード・プロシージャー
- またはバッチ SQL

構文:

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

関数引き数:

表 101. SQLMoreResults 引き数

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

使用法:

この関数は、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_ROW_ARRAY_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() のときに複数の SQL ステートメントがまとめてバッチ化されます。

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可能であ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可能であれば、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、DB2 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 102. 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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「*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*」の『CLI アプリケーションからのストアード・プロシージャーの呼び出し』

関連資料:

- 158 ページの『SQLFreeHandle 関数 (CLI) - ハンドル・リソースの解放』
- 160 ページの『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)』
- 263 ページの『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』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『spcall.c -- Call individual stored procedures』
- 『spclient.c -- Call various stored procedures』
- 『spclires.c -- Contrast stored procedure multiple result set handling methods』

SQLNativeSql 関数 (CLI) - ネイティブ SQL テキストの取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLNativeSql() は、 DB2 CLI がベンダー・エスケープ文節を解釈する方法を表示するために使用します。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・エスケープ文節シーケンス列が入っていた場合、 DB2 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・エスケープ文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。)

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 103. SQLNativeSql 引き数

データ・タイプ	引き数	使用法	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLCHAR *	*InStatementText*	入力	入力 SQL ストリング。
SQLINTEGER	*TextLength1*	入力	*InStatementText* 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 の数 (この関数の Unicode 版の場合は SQLWCHAR の数) が *BufferLength* より大きい場合、*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:

DB2 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したいとき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・エスケープ文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 DB2 CLI は、ベンダーのエスケープ文節の構文エラーしか検出できません。 DB2 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 104. 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。引き数 *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	引き数の値が無効です。	引き数 *InStatementText* は NULL ポインターです。 引き数 *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。	引き数 *TextLength1* は 0 より小さく、SQL_NTS と等しくありませんでした。 引き数 *BufferLength* は、0 より小さい値でした。

制限:

なし。

関連概念:

- 「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」の『CLI アプリケーションでのベンダー・エスケープ文節』
- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 「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」の『CLI 関数戻りコード』

SQLNumParams 関数 (CLI) - SQL ステートメント内のパラメーター数の取得

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLNumParams() は、 SQL ステートメント内のパラメーター・マーカー数を戻します。

構文:

```
SQLRETURN   SQLNumParams (
               SQLHSTMT          StatementHandle,      /* hstmt */
               SQLSMALLINT       *ParameterCountPtr);  /* pcpar */
```

関数引き数:

表 105. SQLNumParams 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLSMALLINT *	*ParameterCountPtr*	出力	ステートメント内のパラメーター数。

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 106. 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。

表 106. SQLNumParams 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可能になりました。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* 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。

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『CLI アプリケーションでのパラメーター・マーカー・バインディング』

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

目的:

仕様:	DB2 CLI 7.x		

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。

構文:

```
SQLRETURN   SQLNextResult (SQLHSTMT StatementHandle1
                           SQLHSTMT StatementHandle2);
```

関数引き数:

表 107. SQLNextResult 引き数

データ・タイプ	引き数	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取り出し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。取り出す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 108. 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 */
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「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」の『CLI アプリケーションからのストアード・プロシージャーの呼び出し』

関連資料:

- 257 ページの『SQLMoreResults 関数 (CLI) - さらに結果セットがあるかどうかの判別』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『spclires.c -- Contrast stored procedure multiple result set handling methods』

SQLNumResultCols 関数 (CLI) - 結果列の数の取得

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLNumResultCols() は、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。

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 */
```

関数引き数:

表 109. SQLNumResultCols 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLSMALLINT *	*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、出力引き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 110. 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可能になりました。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。
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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

- 「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」の『CLI アプリケーションにおける結果セットの用語』

関連資料:

- 11 ページの『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』
- 19 ページの『SQLBindFileToCol 関数 (CLI) - LOB 列への LOB ファイル参照のバインド』
- 93 ページの『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 272 ページの『SQLPrepare 関数 (CLI) - ステートメントの準備』
- 298 ページの『SQLSetColAttributes 関数 (CLI) - 列属性の設定』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『spclient.c -- Call various stored procedures』
- 『spclires.c -- Contrast stored procedure multiple result set handling methods』
- 『tbread.c -- How to read data from tables』

SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

SQLParamData() は、長いデータを分割して送るために SQLPutData() と組み合わせて使用します。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。

構文:

```
SQLRETURN   SQLParamData (
               SQLHSTMT         StatementHandle, /* hstmt */
               SQLPOINTER       *ValuePtrPtr );  /* prgbValue */
```

関数引き数:

表 111. SQLParamData 引き数

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

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 回または複数回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()

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 112. SQLParamData SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換です。	DB2 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 112. 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* 引き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() から引き数 *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可能になりました。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* 引き数が無効でした。
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 112. 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);
```

関連概念:

- 「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」の『CLI でのハンドル』
- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションでのラージ・オブジェクトの使用』
- 「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」の『CLI アプリケーションでのバルク挿入およびバルク更新用の長いデータ』

関連資料:

- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 293 ページの『SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し』
- 27 ページの『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』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』

SQLParamOptions 関数 (CLI) - パラメーターの入力配列の指定

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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);
```

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』

SQLPrepare 関数 (CLI) - ステートメントの準備

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLPrepare() は、提供された入力ステートメント・ハンドルに SQL ステートメントを関連付けます。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 (?) 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。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 113. SQLPrepare 引き数

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

表 113. SQLPrepare 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLINTEGER	*TextLength*	入力	*StatementText* 引き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・エスケープ文節シーケンス列が入っている場合、 DB2 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・エスケープ文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、 DB2 CLI がどのベンダー・エスケープ文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、以下を使用します。

- SQLDescribeParam()
- SQLNumParams()

注: SQLNumParams() 以外の上記の関数を初めて呼び出した場合はすべて、据え置き準備が使用可能になっていれば、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 “?” 文字で表されますが、これを使って、 SQLExecute() の呼び出し時にアプリケーション提供の値をステートメント内のどの位置で置き換えればよいかを指示します。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 114. SQLPrepare SQLSTATE

SQLSTATE	説明	解説
01504	UPDATE または DELETE ステートメントに、WHERE 文節がありません。	*StatementText* の UPDATE ステートメントまたは DELETE ステートメントに、 WHERE 文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	引き数 *StatementText* の WHERE 文節に 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。

表 114. SQLPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
22025	無効なエスケープ・シーケンス	引き数 *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
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、 *StatementText* に含まれている SQL ステートメントの実行がこの許可 ID に許可され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 文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可能になりました。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 114. SQLPrepare SQLSTATE (続き)

SQLSTATE	説明	解説
HY009	引き数の値が無効です。	*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。	引き数 *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。

注:

a xxx は、そのクラス・コードの任意の SQLSTATE を表します。たとえば、 **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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『Unicode 関数 (CLI)』

- 「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」の『DB2 CLI 用の SQLSTATES』

関連タスク:

- 「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」の『CLI アプリケーションでの SQL ステートメントの準備と実行』

関連資料:

- 23 ページの『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』
- 61 ページの『SQLColAttribute 関数 (CLI) - 列属性を戻す』
- 93 ページの『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』
- 97 ページの『SQLDescribeParam 関数 (CLI) - パラメーター・マーカーの記述を戻す』
- 109 ページの『SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 261 ページの『SQLNumParams 関数 (CLI) - SQL ステートメント内のパラメーター数の取得』
- 266 ページの『SQLNumResultCols 関数 (CLI) - 結果列の数の取得』
- 321 ページの『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』
- 「*SQL リファレンス 第 2 巻*」の『COMMIT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『ROLLBACK ステートメント』
- 27 ページの『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』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『spserver.c -- Definition of various types of stored procedures』
- 『tbmod.c -- How to modify table data』

SQLPrimaryKeys 関数 (CLI) - 表の主キー列の取得

目的:

仕様:	DB2 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 115. SQLPrimaryKeys 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CatalogName*	入力	3 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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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() は、シングル表から主キー列を戻します。どの引き数の指定にも、検索パターンは使用はできません。

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 の順序で含まれています。

多くの場合 SQLPrimaryKeys() への呼び出しは、システム・カタログに対して複雑で、それゆえに費用のかかる照会へとマップされるので、その使用を控える必要があります。そして呼び出しを繰り返すよりも、結果を保管しておく方が良いでしょう。

スキーマ名を指定しないと、現行接続で現在有効なものがデフォルトのスキーマ名になります。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) 分の余地を確保しておくか、または接続している DBMS がサポートしている TABLE_CAT、 TABLE_SCHEM、 TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別するために、 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() を呼び出すことができます。

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: DB2 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 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_SUCCESS

- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 116. 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可能になりました。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。	名前の長さ引き数のうちの 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。	DB2 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することができます。

制限:

なし。

例:

```
/* get the primary key columns of a table */
cliRC = SQLPrimaryKeys(hstmt, NULL, 0, tbSchema, SQL_NTS, tbName, SQL_NTS);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』

- 「*管理ガイド: プランニング*」の『主キー』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 151 ページの『SQLForeignKeys 関数 (CLI) - 外部キー列のリストの取得』
- 343 ページの『SQLStatistics 関数 (CLI) - 基本表の索引情報および統計情報の取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbconstr.c -- How to work with constraints associated with tables』

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

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLProcedureColumns() 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 117. SQLProcedureColumns 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CatalogName*	入力	3 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 分割の命名法をサポートしていない (ワークステーション用の DB2 UDB バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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	スキーマ名別に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。 DB2 for MVS/ESA V 4.1 以上の場合、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* 引き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 Universal Database の場合、*SchemaName* には有効なパターン値を入れることができます。有効な検索パターンの詳細は、カタログ関数の入力引き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	プロシージャー名別に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 *	*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 を入れるバッファー。この引き数を使用するのは、 ProcName または SchemaName (あるいはその両方) を空ではない値とするという指定によってすでに制約を受け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() 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認識する必要があります。

ほとんどの場合、 SQLProcedureColumns() への呼び出しはシステム・カタログに対して複雑で、それゆえに費用のかかる照会へとマップされるので、使用を節約する必要があります。それで呼び出しを繰り返すよりも、結果を保管しておく方が良いでしょう。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) の余地を確保しておくか、または 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 MVS/ESA V4.1 から V6 まで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。 V7 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。

DB2 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。

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 を収めたスキーマの名前。 (これも、DB2 for MVS/ESA V 4.1 以降の 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)
この行に関連したタイプ情報を識別します。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)
DATA_TYPE 列の値が文字ストリングまたはバイナリー・ストリングを示すものである場合、この列には SQLCHAR または SQLWCHAR エレメント数を単位とする最大長が入れられます。 GRAPHIC (DBCS) ストリングの場合、これはパラメーターの SQLCHAR または SQLWCHAR エレメントの数になります。

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場合に値を表示するの必要な 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、これは結果表の NUM_PREC_RADIX 列の値に基づいて、列に許可されている合計桁数または合計ビット数のいずれかです。

データ・タイプ精度の表も参照してください。

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 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 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 が厳密な数データ・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ときに使用できます。

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 (DB2 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 (DB2 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き数の元の位置です。左端の引き数の元の位置は、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。)

注: DB2 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 SQLProcedureColumns() 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 118. 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。

表 118. SQLProcedureColumns 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可能になりました。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。	名前の長さ引き数のうちの 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:

SQLProcedureColumns() 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() 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 */
```

関連概念:

- 「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」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 「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」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 288 ページの『SQLProcedures 関数 (CLI) - プロシージャー名のリストの取得』
- 366 ページの『接続属性 (CLI) リスト』
- 427 ページの『データ・タイプ精度 (CLI) 表』
- 428 ページの『データ・タイプ・スケール (CLI) 表』
- 429 ページの『データ・タイプ長 (CLI) 表』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『spcall.c -- Call individual stored procedures』

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

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLProcedures() 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。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 119. SQLProcedures の引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*CatalogName*	入力	3 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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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	スキーマ名別に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。 DB2 for MVS/ESA V 4.1 以上の場合、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* 引き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 Universal Database の場合、*SchemaName* には有効なパターン値を入れることができます。有効な検索パターンの詳細は、カタログ関数の入力引き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() 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。

ほとんどの場合、SQLProcedures() への呼び出しはシステム・カタログに対して複雑で、それゆえに費用のかかる照会へとマップされるので、節約して使用する必要があります。それで呼び出しを繰り返すよりも、結果を保管しておく方が良いでしょう。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) 分の余地を確保しておくか、または接続している DBMS がサポートしている TABLE_CAT、 TABLE_SCHEM、 TABLE_NAME、および COLUMN_NAME 列の実際の長さを判別するために、 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() を呼び出すことができます。

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。

DB2 for MVS/ESA V4.1 から V6 までの間のサーバーにストアード・プロシージャーがある場合、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。 V7 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。

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) 非 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す。つまり、そのオブジェクトには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す。つまり、そのオブジェクトには戻り値があります。

DB2 CLI は常に SQL_PT_PROCEDURE を戻します。

注: DB2 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_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 120. 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。
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 120. SQLProcedures 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可能になりました。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。	名前の長さ引き数のうちの 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:

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。

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 281 ページの『SQLProcedureColumns 関数 (CLI) - プロシージャーの入力/出力パラメーター情報の取得』
- 366 ページの『接続属性 (CLI) リスト』

- 「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」の『CLI 関数戻りコード』

SQLPutData 関数 (CLI) - パラメーターのデータ値の引き渡し

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

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。この関数を使用して、大きなパラメーター値を分割して送ることができます。

構文:

```
SQLRETURN   SQLPutData      (
               SQLHSTMT         StatementHandle,   /* hstmt */
               SQLPOINTER       DataPtr,           /* rgbValue */
               SQLINTEGER       StrLen_or_Ind);    /* cbValue */
```

関数引き数:

表 121. SQLPutData 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
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。
SQLINTEGER	*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。 DB2 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。 DB2 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 DB2 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()

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 122. 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。
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可能になりました。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	引き数の値が無効です。	引き数 *DataPtr* が NULL ポインターで、引き数 *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。	引き数 *DataPtr* は NULL ポインターではなく、引き数 *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・プロシージャー引き数ではデフォルト値はサポートされないので、StrLen_or_Ind 引き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 引き数* 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);
```

関連概念:

- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのバルク挿入およびバルク更新用の長いデータ』

関連資料:

- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 259 ページの『SQLNativeSql 関数 (CLI) - ネイティブ SQL テキストの取得』
- 268 ページの『SQLParamData 関数 (CLI) - データ値が必要な次のパラメーターの取得』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dtlob.c -- How to read and write LOB data』

SQLRowCount 関数 (CLI) - 行カウントの取得

目的:

仕様:	DB2 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実行された 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。

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

構文:

```
SQLRETURN   SQLRowCount (
               SQLHSTMT     StatementHandle,   /* hstmt */
               SQLINTEGER   *RowCountPtr);     /* pcrow */
```

関数引き数:

表 123. SQLRowCount 引き数

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

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実行されたものが 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 *RowCountPtr* の内容を -1 に設定します。

そのステートメントによって影響を受けた可能性がある他表の行 (たとえば、カスケード削除) 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 124. 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。

表 124. SQLRowCount SQLSTATE (続き)

SQLSTATE	説明	解説
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。

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 113 ページの『SQLExecDirect 関数 (CLI) - ステートメントの直接実行』
- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 「*SQL リファレンス 第 1 巻*」の『INSERT スカラー関数』
- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』
- 「*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*」の『CLI 関数戻りコード』

SQLSetColAttributes 関数 (CLI) - 列属性の設定

使用すべきでない:

注:

ODBC 3.0 では SQLSetColAttributes() は使用すべきでない関数なので、DB2 CLI はこの関数をサポートしていません。

現在では、DB2 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。

関連概念:

- 「*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*」の『CLI アプリケーションでの据え置き準備』

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』

SQLSetConnectAttr 関数 (CLI) - 接続属性の設定

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLSetConnectAttr() は、接続の各局面を管理する属性を設定します。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 125. SQLSetConnectAttr 引き数

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

表 125. SQLSetConnectAttr 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、この引き数の長さは **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 DB2 CLI 属性の場合、アプリケーションは *StringLength* 引き数を設定して、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 SQLCHAR エレメントの数 (またはこの関数の Unicode 版の場合は SQLWCHAR エレメントの数) 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 *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。

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 DB2 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、DB2 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* 引き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* 引き数は無視されます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

DB2 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 126. 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。	DB2 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。)

表 126. SQLSetConnectAttr SQLSTATE (続き)

SQLSTATE	説明	解説
08002	接続が使用中です。	引き数 *Attribute* は SQL_ATTR_ODBC_CURSORS で、 DB2 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、DB2 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。	引き数 *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	引き数の値が無効です。	*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。	引き数 *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。
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。 (DB2 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、 DB2 CLI は *ValuePtr* に指定されている値を確認する必要があります。) *Attribute* 引き数は 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* 引き数は 0 より小さい値でしたが、SQL_NTS ではありませんでした。

表 126. SQLSetConnectAttr SQLSTATE (続き)

SQLSTATE	説明	解説
HY092	オプション・タイプが範囲外です。	引き数 *Attribute* に指定された値が、このバージョンの DB2 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。	引き数 *Attribute* に指定された値は、このバージョンの DB2 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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 164 ページの『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』
- 243 ページの『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 366 ページの『接続属性 (CLI) リスト』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『tbread.c -- How to read data from tables』
- 『tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement』

SQLSetConnection 関数 (CLI) - 接続ハンドルの設定

目的:

仕様:	DB2 CLI 2.1		

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 DB2 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。

構文:

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

関数引き数:

表 127. SQLSetConnection 引き数

データ・タイプ	引き数	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:

DB2 CLI バージョン 1 では、DB2 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 DB2 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 DB2 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、DB2 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 DB2 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 DB2 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 128. 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() から引き数 *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);
```

関連概念:

- 「*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*」の『組み込み SQL と DB2 CLI の混合に関する考慮事項』
- 「*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*」の『CLI でのハンドル』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションでのマルチサイト更新 (2 フェーズ・コミット)』

関連資料:

- 83 ページの『SQLConnect 関数 (CLI) - データ・ソースへの接続』
- 103 ページの『SQLDriverConnect 関数 (CLI) - データ・ソースへの (拡張) 接続』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbmconx.c -- How to use multiple databases with embedded SQL.』

SQLSetConnectOption 関数 (CLI) - 接続オプションの設定

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、Unicode 関数 (CLI) を参照してください。

新しい関数への移行

たとえば、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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』

関連資料:

- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』

SQLSetCursorName 関数 (CLI) - カーソル名の設定

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLSetCursorName() は、ステートメント・ハンドルにカーソル名を関連付けます。DB2 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 129. SQLSetCursorName 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CursorName*	入力	カーソル名。
SQLSMALLINT	*NameLength*	入力	*CursorName* 引き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:

DB2 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。 DB2 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することをお勧めします。

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、18 バイト以下です。 18 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 18 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を含むカーソル名 (各国語セットや 2 バイト文字セットの文字など) 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 130. SQLSetCursorName SQLSTATE

SQLSTATE	説明	解説
34000	カーソル名が無効です。	引き数 *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。 引き数 *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 引き数のある `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	引き数の値が無効です。	*CursorName* は、NULL ポインターでした。
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。

表 130. SQLSetCursorName 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。	引き数 *NameLength* 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。

許可:

なし。

例:

```
/* set the name of the cursor */
rc = SQLSetCursorName(hstmtSelect, (SQLCHAR *)"CURSNAME", SQL_NTS);
```

関連概念:

- 「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」の『CLI でのハンドル』
- 「アプリケーション開発ガイド クライアント・アプリケーションのプログラミング」の『各国語サポートとアプリケーション開発に関する考慮事項』
- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションのカーソル』

関連タスク:

- 「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」の『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 「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」の『CLI アプリケーションでのデータの更新と削除』

関連資料:

- 168 ページの『SQLGetCursorName 関数 (CLI) - カーソル名の取得』
- 「アプリケーション開発ガイド クライアント・アプリケーションのプログラミング」の『DBCS 文字セット』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『tbmod.c -- How to modify table data』

SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLSetDescField() は、記述子レコードの単一フィールドの値を設定します。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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);
```

関数引き数:

表 131. SQLSetDescField 引き数

データ・タイプ	引き数	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* 引き数は、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* 引き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* 引き数の値に応じて、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。

表 131. SQLSetDescField 引き数 (続き)

データ・タイプ	引き数	使用法	説明
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、この引き数は **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* 引き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 SQLCHAR エレメントの数 (またはこの関数の Unicode 版の場合は SQLWCHAR エレメントの数) 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* 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* 引き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、たとえば 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 を迅速に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、DB2 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、ステートメントの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 つまたは複数 (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 132. 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 DB2 CLI がサポートしていなかったか、 **ValuePtr* が無効だったため、DB2 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。)

表 132. SQLSetDescField SQLSTATE (続き)

SQLSTATE	説明	解説
07009	記述子索引が無効です。	*FieldIdentifier* 引き数はヘッダー・フィールドでしたが、*RecNumber* 引き数が 0 ではありませんでした。 *RecNumber* 引き数が 0 で、*DescriptorHandle* は IPD でした。 *RecNumber* 引き数は 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、DB2 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* 引き数は IRD に関連付けられていましたが、*FieldIdentifier* 引き数が 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。	TYPE フィールド、あるいは記述子内で TYPE フィールドに関連付けられた他のフィールドが有効ではないか整合性がありません。TYPE フィールドが有効な DB2 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。
HY091	記述子フィールド ID が無効です。	*FieldIdentifier* 引き数に指定された値が、 DB2 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* 引き数に指定されました。 *FieldIdentifier* 引き数が SQL_DESC_ALLOC_TYPE でした。
HY092	オプション・タイプが範囲外です。	*Attribute* 引き数に指定された値が有効ではありませんでした。
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* 引き数』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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションの記述子』
- 「*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*」の『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 11 ページの『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』
- 179 ページの『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 184 ページの『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 315 ページの『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』
- 27 ページの『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』
- 414 ページの『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 401 ページの『記述子 FieldIdentifier 引き数の値 (CLI)』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』

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

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

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。

構文:

```
SQLRETURN   SQLSetDescRec    (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       Type,
                              SQLSMALLINT       SubType,
                              SQLINTEGER        Length,
                              SQLSMALLINT       Precision,
                              SQLSMALLINT       Scale,
                              SQLPOINTER        DataPtr,
                              SQLINTEGER        *StringLengthPtr,
                              SQLINTEGER        *IndicatorPtr);
```

関数引き数:

表 133. SQLSetDescRec 引き数

データ・タイプ	引き数	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。この引き数は、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。
SQLINTEGER	*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。
SQLINTEGER *	*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。
SQLINTEGER *	*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* 引き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・ハンドル SQL_ATTR_USE_BOOKMARKS は、必ず、 *RecNumber* 引き数を 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 134. 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 134. SQLSetDescRec SQLSTATE (続き)

SQLSTATE	説明	解説
07009	記述子索引が無効です。	*RecNumber* 引き数は 0 に設定されており、 *DescriptorHandle* は IPD ハンドルでした。 *RecNumber* 引き数は 0 未満でした。 *RecNumber* 引き数はデータ・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* 引き数は APD、IPD、または ARD でした。 *RecNumber* 引き数は 0 と等しく、 *DescriptorHandle* 引き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、DB2 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* は、非同期で実行中の関数が呼び出された *StatementHandle* と関連しており、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* 引き数は、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。

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);
```

関連概念:

- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションの記述子』
- 「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」の『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 11 ページの『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』
- 179 ページの『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 184 ページの『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 309 ページの『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 27 ページの『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』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』

SQLSetEnvAttr 関数 (CLI) - 環境属性の設定

目的:

仕様:	DB2 CLI 2.1		ISO CLI

SQLSetEnvAttr() は、現行環境の環境属性を設定します。

構文:

```
SQLRETURN   SQLSetEnvAttr    (SQLHENV          EnvironmentHandle, /* henv */
                              SQLINTEGER       Attribute,
                              SQLPOINTER       ValuePtr,          /* Value */
                              SQLINTEGER       StringLength);
```

関数引き数:

表 135. SQLSetEnvAttr 引き数

データ・タイプ	引き数	使用法	説明
SQLHENV	*EnvironmentHandle*	入力	環境ハンドル。
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	指定したい *Attribute* の値。
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、DB2 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 136. 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* 引き数は 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、DB2 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、引き数 *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);
```

関連概念:

- 「*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*」の『DB2 CLI 用の SQLSTATES』

関連資料:

- 197 ページの『SQLGetEnvAttr 関数 (CLI) - 現行の環境属性値の検索』
- 361 ページの『環境属性 (CLI) リスト』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『cli_info.c -- How to get and set environment attributes at the client level』
- 『spcall.c -- Call individual stored procedures』

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

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

等価の関数: SQLBindParameter()

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き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* 引き数を持ちません。 SQLSetParam() を呼び出すことは、*InputOutputType* 引き数を SQL_PARAM_INPUT に設定し、 *BufferLength* 引き数を 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。

新しい関数への移行

たとえば、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);
```

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』
- 27 ページの『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』

SQLSetPos 関数 (CLI) - 行セット (Rowset) 内のカーソル位置の設定

目的:

仕様:	**DB2 CLI** 5.0	**ODBC** 1	

SQLSetPos() は、行セットでカーソル位置を設定します。

構文:

```
SQLRETURN   SQLSetPos (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLUSMALLINT      RowNumber,        /* irow */
               SQLUSMALLINT      Operation,        /* fOption */
               SQLUSMALLINT      LockType);        /* fLock */
```

関数引き数:

表 137. SQLSetPos 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	StatementHandle	入力	ステートメント・ハンドル。
SQLUSMALLINT	*RowNumber*	入力	*Operation* 引き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、 323 ページのRowNumber 引き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 DB2 CLI でもサポートされています。 • SQL_ADD DB2 CLI は、SQLSetPos() 呼び出しで SQL_ADD をサポートしていますが、 *Operation* 引き数を SQL_ADD に設定して SQLBulkOperations() を使用するようお勧めします。

表 137. SQLSetPos 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLUSMALLINT	*LockType*	入力	*Operation* 引き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、DB2 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、 325 ページのLockType 引き数を参照してください。

使用法:

RowNumber 引き数

RowNumber 引き数は行セットの行数を指定し、それに対して *Operation* 引き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* 引き数は 1 ベースです。たとえば、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数 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。以下の操作では、カーソル位置が必要になります。
- 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
- 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 引き数

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

DB2 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

DB2 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。 DB2 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、行ごとのバインドおよび列ごと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、現行の取り出し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、次の取り出し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、それらの行は次の取り出し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

DB2 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* 引き数) 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* 引き数) 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。詳細については、 326 ページの状況および操作配列を参照してください。

SQL_DELETE

DB2 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、以下のものはこの行に関し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。詳細については、326 ページの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 DB2 CLI でもサポートされています。しかし、*Operation* 引き数を SQL_ADD に設定して、 SQLBulkOperations() を使用するようお勧めします。

LockType 引き数

LockType 引き数により、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* 引き数の SQL_LOCK_NO_CHANGE 値だけをサポートします。

LockType 引き数は、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* 引き数を定義します。 DB2 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 138. 操作値

LockType 引き数	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	DB2 CLI ではサポートされない。行を排他的にロックする。

表 138. 操作値 (続き)

LockType 引き数	ロック・タイプ
SQL_LOCK_UNLOCK	DB2 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 139. 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* 引き数は SQL_REFRESH で、データ・タイプ SQL_C_CHAR または SQL_C_BINARY の 1 つまたは複数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す。
01S01	行にエラーがあります。	*RowNumber* 引き数は 0 で、 *Operation* 引き数で指定した操作の実行中に、 1 つまたは複数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 つまたはそれ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。)

表 139. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
01S07	小数点以下切り捨てです。	*Operation* 引き数は 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 つまたは複数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。	引き数 *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。	引き数 *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。	引き数 *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。 引き数 *Operation* は SQL_REFRESH で、 1 つまたは複数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。	引き数 *Operation* は SQL_UPDATE で、結果セットの列への日付またはタイム・スタンプ値の割り当てで、年、月、または日フィールドの範囲が超過しました。 引き数 *Operation* は SQL_REFRESH で、 1 つまたは複数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の範囲が超過した可能性があります。
22008	日時フィールドがオーバーフローしました。	*Operation* 引き数は SQL_UPDATE で、結果セットの列に送られるデータの日時計算で、結果の日時フィールド (年、月、日、時、分、または秒フィールド) が許容範囲を超えたか、または、グレゴリオ歴に基づく日時の法則に対して無効な値になりました。 *Operation* 引き数は SQL_REFRESH で、結果セットから検索されるデータの日時計算で、結果の日時フィールド (年、月、日、時、分、または秒フィールド) が許容範囲を超えたか、または、グレゴリオ歴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。

表 139. SQLSetPos 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可能になりました。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。
HY090	ストリングまたはバッファーの長さが無効です。	*Operation* 引き数は 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* 引き数は 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。

表 139. SQLSetPos SQLSTATE (続き)

SQLSTATE	説明	解説
HY092	オプション・タイプが範囲外です。	*Operation* 引き数は 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。	引き数 *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* 引き数は SQL_UPDATE、SQL_DELETE、または SQL_REFRESH で、 *RowNumber* 引き数で識別する行は、削除されているか、取り出されていません。 *RowNumber* 引き数は 0 で、*Operation* 引き数は SQL_POSITION でした。
HYC00	ドライバーが使用できません。	DB2 CLI またはデータ・ソースは、 *Operation* 引き数または *LockType* 引き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);
```

関連概念:

- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションのカーソル』
- 「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」の『CLI アプリケーションにおける結果セットの用語』

関連タスク:

- 「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」の『CLI アプリケーションでの列方向バインドを使用した配列データの取り出し』
- 「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」の『CLI アプリケーションでの行方向バインドを使用した配列データの取り出し』

関連資料:

- 133 ページの『SQLFetch 関数 (CLI) - 次の行の取り出し』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 50 ページの『SQLBulkOperations 関数 (CLI) - 行のセットの追加、更新、削除、または取り出し』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbread.c -- How to read data from tables』

SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLSetStmtAttr() は、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。

使用可能な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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 140. SQLSetStmtAttr 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLINTEGER	*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 140. SQLSetStmtAttr 引き数 (続き)

データ・タイプ	引き数	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 DB2 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 DB2 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 DB2 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 DB2 CLI からエラー (SQLSTATE HY024) が戻される可能性もあります。

表 140. SQLSetStmtAttr 引き数 (続き)

データ・タイプ	引き数	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 DB2 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 SQLCHAR エレメントの数 (またはこの関数の Unicode 版の場合は SQLWCHAR エレメントの数) 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 *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、

DB2 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。たとえば、DB2 CLI は純キー・セット・カーソルをサポートします。それゆえ、DB2 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 DB2 CLI は、**ValuePtr* 引き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。たとえば、SQLSetStmtAttr() の *ValuePtr* 引き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 つまたは複数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、記述子ハンドルを取り出す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 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* 引き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 141. 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 142. 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。	DB2 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、 DB2 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、DB2 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。

表 142. SQLSetStmtAttr 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。
HY009	引き数の値が無効です。	*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。
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* 引き数が SQL_ATTR_IMP_ROW_DESC または SQL_ATTR_IMP_PARAM_DESC でした。 *Attribute* 引き数は 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。 (DB2 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* 引き数は 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。	引き数 *Attribute* に指定された値が、このバージョンの DB2 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。	引き数 *Attribute* に指定された値は、このバージョンの DB2 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);
```

関連概念:

- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションの記述子』

関連資料:

- 7 ページの『SQLAllocHandle 関数 (CLI) - ハンドルの割り振り』
- 56 ページの『SQLCancel 関数 (CLI) - ステートメントの取り消し』
- 160 ページの『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)』
- 164 ページの『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』
- 243 ページの『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 309 ページの『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 381 ページの『ステートメント属性 (CLI) のリスト』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『dbuse.c -- How to use a database』
- 『tbread.c -- How to read data from tables』

SQLSetStmtOption 関数 (CLI) - ステートメント・オプションの設定

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

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。

新しい関数への移行

たとえば、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);
```

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』

SQLSpecialColumns 関数 (CLI) - 特殊な (行 ID) 列の取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLSpecialColumns() は、表のユニークな行の ID 情報 (主キーまたはユニーク索引) 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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 143. SQLSpecialColumns 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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 143. SQLSpecialColumns 引き数 (続き)

データ・タイプ	引き数	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、次のうちの 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。 以下のうちの 1 つ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:

表の任意の行をユニークに識別する方法が複数ある場合 (つまり、指定した表に複数のユニークな索引がある場合)、 DB2 CLI 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き数を指定しないと、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。 340 ページの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。

ほとんどの場合、SQLSpecialColumns() への呼び出しは、システム・カタログに対して複雑で、それゆえに費用のかかる照会へとマップされるので、使用を控える必要があります。それで呼び出しを繰り返すよりも、結果を保管しておく方が良いでしょう。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) の余地を確保しておくか、または 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。

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* 引き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、 338 ページの表 143 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の必要な 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、これは結果表の NUM_PREC_RADIX 列の値に基づいて、列に許可されている合計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、他の非 IBM RDBMS サーバーから次の値を受け取ることができます。

- 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 144. 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可能になりました。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	引き数の値が無効です。	*TableName* が NULL です。

表 144. 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。	長さ引き数のうちの 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。 長さ引き数のうちの 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);
```

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『分離レベル』
- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』

- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』
- 「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」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 76 ページの『SQLColumns 関数 (CLI) - 表の列の情報の取得』
- 343 ページの『SQLStatistics 関数 (CLI) - 基本表の索引情報および統計情報の取得』
- 353 ページの『SQLTables 関数 (CLI) - 表の情報の取得』
- 427 ページの『データ・タイプ精度 (CLI) 表』
- 428 ページの『データ・タイプ・スケール (CLI) 表』
- 429 ページの『データ・タイプ長 (CLI) 表』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『tbconstr.c -- How to work with constraints associated with tables』

SQLStatistics 関数 (CLI) - 基本表の索引情報および統計情報の取得

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLStatistics() は、指定された表の索引情報を取り出します。また、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。情報は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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 145. SQLStatistics 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*CatalogName*	入力	3 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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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。
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。値は現行値でない可能性がありますが、その値を必ず最新の値にするための試みは行われません。

使用法:

SQLStatistics() は、次の 2 つのタイプの情報を戻します。

- 表の統計情報 (使用可能な場合)。
  - 以下に説明されている結果セットの TYPE 列を、SQL_TABLE_STAT、表の中の行数、およびその表を保管するためのページ数に設定するとき。
  - この結果セットの TYPE 列が索引、索引内のユニークな値の個数、および索引を保管するのに使用されるページ数を示すとき。
- 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。結果セット列は、次に示す順序で SQLStatistics で戻される列に示されています。結果セット内の行は、NON_UNIQUE、TYPE、INDEX_QUALIFIER、INDEX_NAME、および KEY_SEQ の順序になります。

ほとんどの場合、SQLStatistics() への呼び出しはシステム・カタログに対して複雑で、それゆえに高価な照会へとマップされるので、使用を節約する必要があります。それで、呼び出しを繰り返すよりも、結果を保管しておくほうが良いでしょう。

表名に関連したスキーマ修飾子引き数を指定しないと、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。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) の余地を確保しておくか、または SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続先の 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。

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 の入ったスキーマの名前。

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。
- 索引値で重複値が使用できる場合、SQL_TRUE。
- 索引値がユニークでなければならない場合、SQL_FALSE。

- この行が SQL_TABLE_STAT (表自体に関する統計情報) 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 - 表自体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 - 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 - 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 - 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またはハッシュ索引以外であることを示します。

列 8 ORDINAL_POSITION (SMALLINT)
名前が INDEX_NAME 列に示される索引内の列の順位。 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で制御可能なものは、データベース統計の正確さです。つまり、統計の最終更新をいつにするかということです。 (たとえば、DB2 Universal Database では、RUNSTATS コマンドの最終実行時にします。) したがって、多くの場合 SQLStatistics() で戻された統計情報のほうが、上述の SQLCA フィールドに入れられた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_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 146. 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可能になりました。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	引き数の値が無効です。	*TableName* が NULL です。

表 146. SQLStatistics 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。	名前の長さ引き数のうちの 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。 名前の長さ引き数のうちの 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することができます。

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);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』

- 「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」の『Unicode 関数 (CLI)』
- 「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」の『DB2 CLI 用の SQLSTATES』
- 「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」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 76 ページの『SQLColumns 関数 (CLI) - 表の列の情報の取得』
- 337 ページの『SQLSpecialColumns 関数 (CLI) - 特殊な (行 ID) 列の取得』
- 「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」の『CLI 関数戻りコード』

関連サンプル:

- 『tbconstr.c -- How to work with constraints associated with tables』
- 『tbinfo.c -- How to get information about tables from the system catalog tables』

SQLTablePrivileges 関数 (CLI) - 表に関連する特権の取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLTablePrivileges() は、表と各表の関連特権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。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、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 */
```

関数引き数:

表 147. SQLTablePrivileges 引き数

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

表 147. SQLTablePrivileges 引き数 (続き)

データ・タイプ	引き数	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 分割の命名法をサポートしていない (DB2 UDB for UNIX および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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	スキーマ名別に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	*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* 入力引き数は、探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 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、列単位で当てはまる場合と当てはまらない場合があります。たとえば、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。他のデータ・ソースの場合は、個々の列が同じ表特権をもっている場合、アプリケーションは `SQLColumnPrivileges()` を呼び出す必要があります。

ほとんどの場合、 `SQLTablePrivileges()` への呼び出しはシステム・カタログに対して複雑で、それゆえに高価な照会へとマップされるので、使用を節約する必要があります。それで、呼び出しを繰り返すよりも、結果を保管しておくほうが良いでしょう。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) の余地を確保しておくか、または SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続先の 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。

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 の入ったスキーマの名前。

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。これは、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
- INDEX
- DELETE
- 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。

これは、「YES」、「NO」、または NULL のいずれかです。

注: DB2 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_SUCCESS

- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 148. 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可能になりました。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。	名前の長さ引き数のうちの 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。 名前の長さ引き数のうちの 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 privileges associated with a table */
cliRC = SQLTablePrivileges(hstmt,
                           NULL,
                           0,
                           tbSchemaPattern,
                           SQL_NTS,
                           tbNamePattern,
                           SQL_NTS);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 288 ページの『SQLProcedures 関数 (CLI) - プロシージャー名のリストの取得』
- 353 ページの『SQLTables 関数 (CLI) - 表の情報の取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbinfo.c -- How to get information about tables from the system catalog tables』

SQLTables 関数 (CLI) - 表の情報の取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLTables() 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返されますが、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取り出すことができます。

同等の Unicode: 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 から Unicode 関数へのマッピングの詳細は、 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 */
```

関数引き数:

表 149. SQLTables 引き数

データ・タイプ	引き数	使用法	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル
SQLCHAR *	*CatalogName*	入力	3 つの部分から成る表名のカタログ修飾子。パターン値 (*pattern value*) が含まれることがあります。ターゲットの DBMS が 3 分割の命名法をサポートしていない (DB2 UDB for Linux、UNIX、および Windows バージョン 8 などの) ときに非空のストリングを指定した場合や、 *NameLength1* が 0 でない場合、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、DB2 UDB for z/OS and OS/390 などの 3 分割命名法をサポートする DBMS の有効なフィルターになります。
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	スキーマ名別に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	*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 149. SQLTables 引き数 (続き)

データ・タイプ	引き数	使用法	説明
SQLCHAR *	*TableType*	入力	表タイプで結果セットを修飾する場合の*値リスト* を入れるバッファー。 値のリストは、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、ALIAS、 HIERARCHY TABLE、 INOPERATIVE VIEW、 NICKNAME、 MATERIALIZED QUERY TABLE、 SYSTEM TABLE、 TABLE、 TYPED TABLE、 TYPED VIEW、または VIEW などがあります。 *TableType* 引き数が 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* 入力引き数は、探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() 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* 引き数用の特殊な方法を適用します。 *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対するコンマで区切った値となります。それぞれの値は単一の引用符でくくるか、あるいはくくらないでおきます。たとえば、″'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() を、*SchemaName*、*TableName*、および *TableType* 引き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。大量の表、ビュー、別名などを含むいくつかのデータ・ソースの場合、このシナリオでは非常に大きな結果セットにマップし、非常に長い取り出し時間がかかります。エンド・ユーザーが長い検索時間を短縮できるように、3 つのメカニズムが導入されています。つまり、3 つの構成キーワード (SCHEMALIST、 SYSCHEMA、TABLETYPE) を CLI 初期設定ファイルに設定しておけば、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() 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 の順序になっています。

ほとんどの場合、SQLTables() への呼び出しはシステム・カタログに対して複雑で、それゆえに高価な照会へとマップされるので、使用を節約する必要があります。それで、呼び出しを繰り返すよりも、結果を保管しておくほうが良いでしょう。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限に従うように、 128 の最大長属性で宣言されています。 DB2 名は 128 より小さいので、アプリケーションは、出力バッファー用に常に 128 文字 (および NULL 終止符文字) の余地を確保しておくか、または SQL_MAX_CATALOG_NAME_LEN、SQL_MAX_OWNER_SCHEMA_LEN、SQL_MAX_TABLE_NAME_LEN、および SQL_MAX_COLUMN_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続先の 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。

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、または同義語の名前。

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。

列

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

診断:

表 150. 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可能になりました。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	引き数の値が無効です。	*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。

表 150. SQLTables 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。	名前の長さ引き数のうちの 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。 名前の長さ引き数のうちの 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);
```

関連概念:

- 「*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*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『DB2 CLI 用の SQLSTATES』
- 「*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*」の『CLI アプリケーションのカタログ関数の入力引き数』

関連資料:

- 76 ページの『SQLColumns 関数 (CLI) - 表の列の情報の取得』
- 349 ページの『SQLTablePrivileges 関数 (CLI) - 表に関連する特権の取得』
- 「*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*」の『CLI 関数戻りコード』

関連サンプル:

- 『tbinfo.c -- How to get information about tables from the system catalog tables』
- 『tbread.c -- How to read data from tables』

SQLTransact 関数 (CLI) - トランザクション管理

使用すべきでない:

注:

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 DB2 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 DB2 CLI プログラムで使用することをお勧めします。

新しい関数への移行

たとえば、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);
```

関連資料:

- 1 ページの『CLI と ODBC 関数のサマリー』
- 109 ページの『SQLEndTran 関数 (CLI) - 接続または環境のトランザクションの終了』

第 2 章 CLI 属性 - 環境、接続、およびステートメント

環境、接続、およびステートメントには、それぞれ DB2 CLI の動作に影響する定義済みの属性の集まりがあります。これらの属性にはデフォルト値がありますが、異なった値を設定することにより DB2 CLI の動作を変更することができます。この章では、DB2 CLI の動作をカスタマイズするのに設定することのできる環境、接続、およびステートメントの属性をリストします。

環境属性 (CLI) リスト

注: 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 DB2 CLI 固有の環境属性を設定することができます。

SQL_ATTR_CONNECTION_POOLING

環境レベルでの CLI 接続のプール処理を使用可能または使用不能にする 32 ビット整数値。 ODBC アプリケーションがこの値を設定すると、CLI レベルではなく、 ODBC Driver Manager での接続プールが使用可能になります。 CLI に直接アクセスするために作成されたアプリケーションだけがこの環境属性を設定して、 CLI 接続プールを制御することができます。使用される値は次のとおりです。

- **SQL_CP_OFF** = 接続のプール処理はオフにされます。これはデフォルトです。
- SQL_CP_ONE_PER_DRIVER = それぞれの DB2 CLI アプリケーションで単一のグローバル接続プールがサポートされています。プール内のすべての接続は、アプリケーションと関連しています。
- SQL_CP_ONE_PER_HENV = それぞれの環境で単一の接続プールがサポートされています。プール内のそれぞれの接続は、1 つの環境と関連しています。

接続のプール処理は、SQLSetEnvAttr() を呼び出して、SQL_ATTR_CONNECTION_POOLING 属性を SQL_CP_ONE_PER_DRIVER または SQL_CP_ONE_PER_HENV に設定することによって使用可能になります。 SQLSetEnvAttr() への呼び出しの環境ハンドルが NULL に設定されているので、 SQL_ATTR_CONNECTION_POOLING がプロセス・レベルの属性になります。接続のプール処理が使用可能になると、アプリケーションは、 *InputHandle* 引き数を SQL_HANDLE_ENV に設定して SQLAllocHandle() を呼び出すことにより、暗黙的に共有環境を割り当てます。

環境の SQL_ATTR_CONNECTION_POOLING 属性を設定するときに NULL の環境ハンドルを指定して SQLSetEnvAttr() を呼び出しているので、接続のプール処理が使用可能になり、アプリケーションの共有環境が選択された後には、この属性をリセットすることはできません。接続のプール処理が共

有環境ですでに使用可能になっていて、この属性が設定されている場合、この属性の影響を受けるのはこの後に割り当てられる共有環境だけです。

SQL_ATTR_CONNECTTYPE

注: これは、SQL_CONNECTTYPE 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。処理を調整する必要がある場合、SQL_ATTR_SYNC_POINT 接続オプションとの組み合わせにおいて、このオプションを考慮に入れる必要があります。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 つのデータベースまたは複数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。

  SQL_ATTR_SYNC_POINT 属性の現行設定は無視されます。

  これはデフォルトです。
- 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 の指定に対応しており、この設定を SQL_ATTR_SYNC_POINT 接続オプションと合わせて考慮する必要があります。上記の 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。

この属性は、接続ハンドルを割り振る前に設定する必要があります。そうでなければ、SQLSetEnvAttr() 呼び出しは拒否されます。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE 値と SQL_ATTR_SYNC_POINT 値は、同じ値でなければなりません。この属性も、SQLSetConnectAttr() 関数を使用して設定できます。アプリケーションが SQL_ATTR_CONNECTTYPE 属性を設定するときに、接続単位ではなく、環境レベルで設定することをお勧めします。ドライバー固有の環境属性の設定のための SQLSetEnvAttr() の呼び出しは ODBC でサポートされていないので、調整 DB2 トランザクションを利用するために書かれた ODBC アプリケーションは、 SQLSetConnectAttr() を使って各接続ごとに接続レベルでこれらの属性を設定する必要があります。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_CP_MATCH

接続を接続プールから選択する方法を決定する 32 ビット値。
SQLConnect() または SQLDriverConnect() が呼び出されると、 DB2 CLI (または使用している場合は ODBC Driver Manager) はどの接続をプールから再利用するかを決定します。 DB2 CLI (または ODBC Driver Manager) は、呼び出し内の接続オプションとアプリケーションが設定した接続属性

を、プール内のキーワードと接続属性に対して突き合わせます。この属性値で、突き合わせ基準の精度レベルが決まります。

次の値を使用して、この属性値を設定します。

- **SQL_CP_STRICT_MATCH** = 呼び出しの接続オプションとアプリケーションが設定した接続属性が正確に一致した接続だけが再利用されます。これはデフォルトです。
- SQL_CP_RELAXED_MATCH = 接続ストリング・キーワードが一致する接続を使用することができます。キーワードは必ず一致しなければなりませんが、すべての接続属性が一致しなければならないわけではありません。

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。

SQL_ATTR_ODBC_VERSION

特定の機能が ODBC 2.x (DB2 CLI v2) または ODBC 3.0 (DB2 CLI v5) の動作を示すかどうかを決定する 32 ビット整数。

すべての DB2 CLI アプリケーションでこの環境属性を設定するようお勧めします。 ODBC アプリケーションでは、SQLHENV 引き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。

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 (DB2 CLI v5) 動作が発生します。
  - DB2 CLI は、日付、時刻、およびタイム・スタンプに、ODBC 3.0 (DB2 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()`、 `SQLGetDiagRec()` が呼び出されると、DB2 CLI は ODBC 3.0 (DB2 CLI v5) SQLSTATE コードを戻します。
  - `SQLTables()` への呼び出しの *CatalogName* 引き数は検索パターンを受け入れます。
- SQL_OV_ODBC2 により、次の ODBC 2.x (DB2 CLI v2) 動作が発生します。
  - DB2 CLI は、日付、時刻、およびタイム・スタンプに ODBC 2.x (DB2 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()`、 `SQLGetDiagRec()` が呼び出されると、DB2 CLI は ODBC 2.0 (DB2 CLI v2) SQLSTATE コードを戻します。
  - `SQLTables()` への呼び出しの *CatalogName* 引き数は検索パターンを受け入れません。

SQL_ATTR_OUTPUT_NTS

出力引き数における NULL 終了の使用を制御する 32 ビット整数値。以下の値を指定することができます。

- **SQL_TRUE**: DB2 CLI は、 NULL 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。

  これはデフォルトです。

- SQL_FALSE: DB2 CLI は、NULL 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

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当て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* 引き数を SQL_NULL_HANDLE に設定する必要があります。この設定は、プロセスの存続期間中はずっと有効です。一般に、この属性が使用されるのは、大量の CLI 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についてだけです。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。

以下の値を組み合わせてビット・マスクを形成することができます。

- 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、DB2 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 接続タイプを 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。これを設定すると、DB2 CLI は、それぞれの関数呼び出しの現行プロセス ID をチェックする必要がなくなります。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_SYNC_POINT

注: これは、SQL_SYNC_POINT に取って代わる属性です。
アプリケーションが 1 フェーズ調整トランザクションと 2 フェーズ調整トランザクションのどちらかを選択するのに使える 32 ビット整数値。以下の値を指定することができます。

- **SQL_TWOPHASE**: 複数のデータベース・トランザクションで、各データベースが行った作業をコミットする場合には、2 フェーズ・コミットが用いられます。このとき、このプロトコルをサポートする複数のデータベース間で 2 フェーズ・コミットを調整するために、トランザクション・マネージャーを使用する必要があります。 1 つのトランザクション内で、複数の読み取り側および複数の更新側があっても許可されます。

注: SQL_ONEPHASE オプションはサポートされなくなりました。 SQL_ONEPHASE を設定しても、 SQL_TWOPHASE オプションの 2 フェーズ動作になります。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE 値と SQL_ATTR_SYNC_POINT 値は、同じ値でなければなりません。この属性も、SQLSetConnectAttr() 関数を使用して設定できます。アプリケーションがこれら 2 つの属性を設定するときに、接続レベルによってではなく、環境レベルで設定することをお勧めします。ドライバー固有の環境属性の設定のための SQLSetEnvAttr() の呼び出しは ODBC でサポートされていないので、調整 DB2 トランザクションを利用するために書かれた ODBC アプリケーションは、 SQLSetConnectAttr() を使って各接続ごとに接続レベルでこれらの属性を設定する必要があります。

注: これは、IBM 定義の拡張機能です。組み込み SQL では、SYNCPOINT NONE という追加の同期点設定があります。 SYNCPOINT NONE は、同じデータベースへの複数の接続を許可していないので、SQL_ATTR_CONNECTTYPE 属性の SQL_CONCURRENT_TRANS 設定よりも限定された設定です。結果的に、DB2 CLI は SYNCPOINT NONE をサポートする必要はありません。

SQL_ATTR_USE_2BYTES_OCTET_LENGTH

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。

SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA

列名がネットワークを介して送信されて、SQLDescribeCol()、SQLColAttribute()、および SQLGetDescField() への呼び出しに対して戻されるかどうかを指定する 32 ビット整数値。以下の値を指定することができます。

- SQL_TRUE: 列名は、ネットワーク・フローには含まれません。列番号だけが戻されます。
- **SQL_FALSE**: 列名は、ネットワーク・フローに含まれます (デフォルト値)。

注: これは、IBM 定義の拡張機能です。

SQL_CONNECTTYPE

この *Attribute* は、362 ページの『SQL_ATTR_CONNECTTYPE』 に置き換えられました。

SQL_MAXCONN

この *Attribute* は、363 ページの『SQL_ATTR_MAXCONN』 に置き換えられました。

SQL_SYNC_POINT

この *Attribute* は、364 ページの『SQL_ATTR_SYNC_POINT』 に置き換えられました。

関連資料:

- 61 ページの『SQLColAttribute 関数 (CLI) - 列属性を戻す』
- 93 ページの『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』

- 170 ページの『SQLGetData 関数 (CLI) - 列からのデータの取得』
- 179 ページの『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 197 ページの『SQLGetEnvAttr 関数 (CLI) - 現行の環境属性値の検索』
- 「*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*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

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 151. 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_ANSI_APP	可	不可	不可
SQL_ATTR_AUTO_IPD (読み取り専用)	不可	可	可
SQL_ATTR_AUTOCOMMIT	可	可	可 [b]
SQL_ATTR_CLISCHEMA	可	可	可
SQL_ATTR_CONN_CONTEXT	可	不可	不可
SQL_ATTR_CONNECT_NODE	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_PACKAGE_PATH	可	可	可
SQL_ATTR_CURRENT_PACKAGE_SET	可	可 [a]	不可 [*]
SQL_ATTR_CURRENT_SCHEMA	可	可	可
SQL_ATTR_DB2_SQLERRP (読み取り専用)	不可	可	可
SQL_ATTR_DB2ESTIMATE	不可	可	可
SQL_ATTR_DB2EXPLAIN	不可	可	可
SQL_ATTR_ENLIST_IN_DTC	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_LOGIN_TIMEOUT	可	不可	不可
SQL_ATTR_LONGDATA_COMPAT	可	可	可
SQL_ATTR_QUIET_MODE	可	可	可
SQL_ATTR_SYNC_POINT	可	不可	不可
SQL_ATTR_TXN_ISOLATION	不可	可 [b]	可 [a]
SQL_ATTR_WCHARTYPE	可	可 [b]	可 [b]

表 151. 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

[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。

[*] 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 (DB2 CLI v2)

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。
- **SQL_MODE_READ_WRITE**: 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。 DB2 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_ANSI_APP (DB2 CLI v7)

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**: 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_AUTO_IPD (DB2 CLI v5)

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 (DB2 CLI v2)

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**: デフォルトでは、DB2 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。

  SQL_AUTOCOMMIT_ON はデフォルト値です。

  注: この値が調整された分散作業接続単位の場合、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_CLISCHEMA (DB2 CLI v6)

使用するホスト DBMS 上に保管されている DB2 ODBC カタログ・ビュ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。

DB2 ODBC カタログは、DB2 Connect を通じてホスト DBMS に接続する ODBC アプリケーションにおいて、表リストのスキーマ呼び出しパフォーマンスを向上するように設計されています。

DB2 ODBC カタログは、ホスト DBMS 上で作成および保守されます。これには、実際の DB2 カタログに定義されているオブジェクトを表す行が入っていますが、各行に入っているのは、ODBC 操作をサポートするのに必要な列だけです。 DB2 ODBC カタログ内の表は、 ODBC アプリケーションでの迅速なカタログ・アクセスをサポートするために事前結合され、明確に索引化されています。

システム管理者は、複数の DB2 ODBC カタログ・ビューを作成し、そのおのおのに個々のユーザー・グループが必要とする行だけを取り込むことがで

きます。各エンド・ユーザーは、自分が使用する DB2 ODBC カタログを (この属性を設定することによって) 選択できます。

DB2 CLI/ODBC ドライバーの構成キーワード CLISCHEMA を使うと、この属性のデフォルト値を指定することができます。

SQL_ATTR_CONN_CONTEXT (DB2 CLI v5)

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、DB2 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接続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。

SQL_ATTR_CONNECT_NODE (DB2 CLI v6)

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ノードを指定する 32 ビットの整数。この設定は、環境変数 DB2NODE の値をオーバーライドします。以下の値に設定できます。

- 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。

これに対応する ConnectNode という DB2 CLI/ODBC 構成キーワードもあります。

SQL_ATTR_CONNECTION_DEAD (DB2 CLI v6)

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。 DB2 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 DB2 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 (DB2 CLI v5)

この接続属性は ODBC により定義されますが、DB2 CLI ではサポートされません。その結果、この属性を設定または取得しようとすると、SQLSTATE が HYC00 (ドライバーが機能しない) になります。

アプリケーションに戻る前に、要求が完了するまで待機する秒数に対応する 32 ビット整数値。

DB2 CLI は、常に *ValuePtr* が 0 (デフォルト値) に設定されたかのように動作します。タイムアウトはありません。

SQL_ATTR_CONNECTTYPE (DB2 CLI v2)

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。処理を調整する必要がある場合、SQL_ATTR_SYNC_POINT 接続オプションとの組み合わせにおいて、このオプションを考慮に入れる必要があります。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 つのデータベースまたは複数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_ATTR_SYNC_POINT オプションの現行設定は無視されます。

  これはデフォルトです。
- 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 の指定に対応しており、この設定を SQL_ATTR_SYNC_POINT 接続オプションと合わせて考慮する必要があります。上記の 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。

このオプションは、接続要求を行う前に設定する必要があります。そうしない場合は、`SQLSetConnectAttr()` 呼び出しは拒否されます。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE 値と SQL_ATTR_SYNC_POINT 値は、同じ値でなければなりません。最初の接続で、以後の接続のために受け入れ可能な属性を判別します。アプリケーションが SQL_ATTR_CONNECTTYPE 属性を設定するときに、接続レベルではなく、環境レベルで設定することをお勧めします。ドライバー固有の環境属性の設定のための `SQLSetEnvAttr()` の呼び出しは ODBC でサポートされていないので、調整 DB2 トランザクションを利用するために書かれた ODBC アプリケーションは、各接続ごとに接続レベルでこれらの属性を設定する必要があります。

CONNECTTYPE DB2 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 (DB2 CLI v8.2)

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_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。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_CURRENT_PACKAGE_SET (DB2 CLI v5)

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。

CLI/ODBC 構成キーワード CURRENTPACKAGESET の設定は、スキーマ名の指定に代わる、代替の方法です。

SQL_ATTR_CURRENT_SCHEMA (DB2 CLI v2)

注: これは、IBM 定義の拡張機能です。
szSchemaName ポインターを NULL に設定する場合に、 SQLColumns() 呼び出しで DB2 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* 引き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。

SQL_ATTR_DB2_SQLERRP (DB2 CLI v6)
sqlca の *sqlerrp* フィールドを収めた NULL 終了ストリングを指す sqlpointer。

製品を示す 3 文字の ID と、その後に製品のバージョン、リリース、修正レベルを示す 5 桁の数字が続きます。たとえば SQL08010 は、DB2 Universal Database バージョン 8 リリース 1 修正レベル 0 を指します。

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。

SQL_ATTR_DB2ESTIMATE (DB2 CLI v2)
オプティマイザーから戻された見積もりを報告するために、 DB2 CLI が SQL 照会の準備の終わりにダイアログ・ウィンドウを表示するかどうかを指定する 32 ビット整数。

- **0**: 見積もりは戻されません。

  これはデフォルトです。

- 非常に大きい正の整数: DB2 CLI がウィンドウをポップアップして見積もりを報告するしきい値。この正の整数値は、PREPARE と関連付けられている SQLCA の SQLERRD(4) フィールドに対して比較されます。SQLERRD(4) フィールドの値が以前設定された正の整数より大きい場合、見積もりウィンドウが表示されます。

  グラフィカル・ウィンドウには、オプティマイザーの評価と、この照会を続行するか取り消すかを選択するためのボタンが表示されます。

  このオプションの推奨値は 60000 です。

このオプションは、SQL_ATTR_QUIET_MODE との組み合わせで使用され、グラフィカル・ユーザー・インターフェースのあるアプリケーションのみ適用されます。アプリケーションは、このオプションを使用しないで、SQLPrepare() の後で SQLGetSQLCA() を呼び出して照会を出し、次いで適切な情報を表示することによってこの機能を直接インプリメントすれば、より統合化された包括的なインターフェースを実現できます。

SQL_ATTR_DB2ESTIMATE 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。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_DB2EXPLAIN (DB2 CLI v2)

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使用不可になっています (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が使用可能、 Explain 表オプション機能が使用不可になっています (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が使用不可、 Explain 表オプション機能が使用可能になっています (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使用可能になっています (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。

DB2EXPLAIN DB2 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。

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_ENLIST_IN_DTC (DB2 CLI v5.2)

以下のいずれかの SQLPOINTER。

- 非 NULL 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、 DB2 CLI/ODBC ドライバーに求めています。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、 DB2 CLI/ODBC ドライバーに求めています。

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_INFO_ACCTSTR (DB2 CLI v6)

DB2 Connect の使用時に、ホスト・データベース・サーバーに送信されるクライアント会計情報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 CLI v6)

DB2 Connect の使用時に、ホスト・データベース・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 (DB2 CLI v8)

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 は、ID を動的 SQL ステートメント・キャッシュに挿入されている任意のステートメントと関連付けます。

この属性は、DB2 UDB for z/OS バージョン 8 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_INFO_USERID (DB2 CLI v6)

DB2 Connect の使用時に、ホスト・データベース・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 CLI v6)

DB2 Connect の使用時に、ホスト・データベース・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保ちます。

- 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。

SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI v2)

この接続属性は ODBC で定義されていますが、DB2 CLI ではサポートされません。デフォルト値以外の値でこの属性を設定しようとすると、 01S02 の SQLSTATE (オプション値が変更されました) が出されます。

ログイン要求の完了を待機し始めてからアプリケーションに戻るまでの秒数に対応する 32 ビット整数値。 *ValuePtr* 引き数で許可されている値は **0** だけです。つまり、接続を試行すると、接続が確立されるか、基礎となる通信層がタイムアウトになるまで待機します。

SQL_ATTR_LONGDATA_COMPAT (DB2 CLI v2)

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、それぞれ 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**: ラージ・オブジェクト・データ・タイプは、それぞれの IBM 定義のタイプ (SQL_BLOB、SQL_CLOB、SQL_DBCLOB) で報告されます。これはデフォルトです。
- 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_METADATA_ID (ODBC 3.0)

この接続属性は ODBC により定義されますが、DB2 CLI ではサポートされません。その結果、この属性を設定または取得しようとすると、SQLSTATE が HYC00 (ドライバーが機能しない) になります。

カタログ関数のストリング引き数を処理する方法を決定する SQLUINTEGER 値。

SQL_ATTR_ODBC_CURSORS (ODBC 2.0)

この接続属性は ODBC により定義されますが、DB2 CLI ではサポートされません。その結果、この属性を設定または取得しようとすると、SQLSTATE が HYC00 (ドライバーが機能しない) になります。

ドライバー・マネージャーが ODBC カーソル・ライブラリーを使用する方法を指定する 32 ビット・オプション。

SQL_ATTR_PACKET_SIZE (ODBC 2.0)

この接続属性は ODBC により定義されますが、DB2 CLI ではサポートされません。その結果、この属性を設定または取得しようとすると、SQLSTATE が HYC00 (ドライバーが機能しない) になります。

ネットワーク・パケット・サイズをバイト単位で指定する 32 ビット整数値。

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 UDB 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 UDB for z/OS スレッドの CORRELATION IDENTIFIER として使用されます。

SQL_ATTR_QUIET_MODE (DB2 CLI v2)

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、 DB2 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。たとえば、エンド・ユーザーが (DB2 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、 DB2 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、DB2 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上記の例では、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、 DB2 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。 DB2 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* 引き数を SQL_DRIVER_NOPROMPT に設定します。)

SQL_ATTR_SYNC_POINT (DB2 CLI v2)

アプリケーションが 1 フェーズ調整トランザクションと 2 フェーズ調整トランザクションのどちらかを選択するのに使える 32 ビット整数値。以下の値を指定することができます。

- **SQL_TWOPHASE**: 複数のデータベース・トランザクションで、各データベースが行った作業をコミットする場合には、2 フェーズ・コミットが

用いられます。このとき、このプロトコルをサポートする複数のデータベース間で 2 フェーズ・コミットを調整するために、トランザクション・マネージャーを使用する必要があります。 1 つのトランザクション内で、複数の読み取り側および複数の更新側があっても許可されます。

注: SQL_ONEPHASE オプションはサポートされなくなりました。 SQL_ONEPHASE を設定しても、 SQL_TWOPHASE オプションの 2 フェーズ動作になります。

アプリケーション内のすべての接続の SQL_ATTR_CONNECTTYPE 値と SQL_ATTR_SYNCPOINT 値は、同じ値でなければなりません。最初の接続で、以後の接続のために受け入れ可能な属性を判別します。アプリケーションが SQL_ATTR_CONNECTTYPE 属性を設定するときに、接続レベルではなく、環境レベルで設定することをお勧めします。ドライバー固有の環境属性の設定のための `SQLSetEnvAttr()` の呼び出しは ODBC でサポートされていないので、調整 DB2 トランザクションを利用するために書かれた ODBC アプリケーションは、接続レベルでこれらの属性を設定する必要があります。

注: これは、IBM 拡張機能です。組み込み SQL では、SYNCPOINT NONE という追加の同期点設定があります。 SYNCPOINT NONE は、同じデータベースへの複数の接続を許可していないので、 SQL_ATTR_CONNECTTYPE オプションの SQL_CONCURRENT_TRANS 設定よりも限定された設定です。結果的に、DB2 CLI は SYNCPOINT NONE をサポートする必要はありません。

SQL_ATTR_TRACE (ODBC 1.0)

この接続属性は ODBC により定義されますが、DB2 CLI ではサポートされません。その結果、この属性を設定または取得しようとすると、 SQLSTATE が HYC00 (ドライバーが機能しない) になります。

トレースを実行するかどうか DB2 CLI に指示する 32 ビット整数値。

この属性を使用する代わりに、TRACE DB2 CLI/ODBC 構成キーワードを使用して DB2 CLI トレース機能を設定することができます。

SQL_ATTR_TRACEFILE (ODBC 1.0)

この接続属性は ODBC により定義されますが、DB2 CLI ではサポートされません。その結果、この属性を設定または取得しようとすると、 SQLSTATE が HYC00 (ドライバーが機能しない) になります。

トレース・ファイルの名前が入っている NULL で終了する文字ストリング。

この属性を使用する代わりに TRACEFILENAME DB2 CLI/ODBC 構成キーワードを使用して、DB2 CLI トレース・ファイル名を設定します。

SQL_ATTR_TRANSLATE_LIB (DB2 CLI v5)

この接続属性は ODBC で定義されていますが、DB2 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。

これは、変換 .DLL に渡される 32 ビット・フラグ値です。

DB2 Client Application Enabler for Windows または Application Development Client for Windows をインストールしたディレクトリーを指示します。 DB2TRANS.DLL は、コード・ページ・マッピング表が入っている DLL です。

SQL_ATTR_TRANSLATE_OPTION (ODBC 1.0)

この接続属性は ODBC で定義されていますが、DB2 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_TXN_ISOLATION (DB2 CLI v2)

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 DB2 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** - 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。これはデフォルトです。
- 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_WCHARTYPE (DB2 CLI v2)

アプリケーションで使用したい 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 データを複数バイト・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可能な 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 (L リテラル、 'wc' ストリング関数など) 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**: 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。

  これはデフォルトです。

注: これは、IBM 定義の拡張機能です。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『分離レベル』
- 「*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*」の『Unicode 関数 (CLI)』
- 「*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*」の『CLI アプリケーションのカーソル』
- 「*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*」の『マルチスレッド CLI アプリケーション』
- 「*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*」の『Unicode CLI アプリケーション』

関連資料:

- 164 ページの『SQLGetConnectAttr 関数 (CLI) - 現在の属性設定の取得』
- 243 ページの『SQLGetStmtAttr 関数 (CLI) - ステートメント属性の現行設定値の取得』
- 298 ページの『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「コマンド・リファレンス」の『BIND コマンド』
- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』
- 「*管理 API リファレンス*」の『sqleBeginCtx - アプリケーション・コンテキストの作成およびアタッチ』
- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『DBCS 文字セット』
- 「*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*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

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ている DB2 CLI または ODBC のバージョンは、下記のとおりです。別のデータ・ソースも利用できるように、さらに多くが今後定義されるものと予想されます。

SQL_ATTR_APP_PARAM_DESC (DB2 CLI v5)

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 (DB2 CLI v5)

ステートメント・ハンドルでの以後の取り出しを行うための 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_BLOCK_FOR_NROWS (DB2 CLI v8)

結果セットの取り出し時に、サーバーから戻されるブロック・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 (DB2 CLI v8)

LOB ブロック・フェッチを有効にするかどうかを指定するブール属性。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 ブロッキング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 データが戻されます。

SQL_ATTR_CALL_RETURN (DB2 CLI v8)

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 (たとえば、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 (DB2 CLI v8)

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。この機能は「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、または使用可能バッファー・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、 DB2 CLI にとって特に意味はありません。単に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機能が有効になります。

SQL_ATTR_CHAINING_END (DB2 CLI v8)

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/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、 DB2 CLI にとって特に意味はありません。単に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機能が無効になります。

SQL_ATTR_CLIENT_LOB_BUFFERING (DB2 CLI v8)

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、同時待機数やネットワーク・フローが少なくなります。この属性に指定できる値は、以下のとおりです。

- SQL_CLIENTLOB_USE_LOCATORS - 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 (DB2 CLI v6)

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。

SQL_ATTR_CLOSEOPEN (DB2 CLI v6)

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連結され、 2 つのステートメントが結合されて (2 つではなく) 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・ステートメントは連結されず、オープン・カーソルがあるとエラーが戻されます。これはデフォルトです。
- 1 = クローズ・ステートメントとオープン・ステートメントを連結します。

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_CONCURRENCY (DB2 CLI v2)

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。転送専用の静的なキー・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。転送専用のキー・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。

静的な転送専用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。

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、DB2 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。

- 転送専用: 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、DB2 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 (DB2 CLI v2)

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取り出せ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 DB2 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 (DB2 CLI v6)

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 SQLExecDirect() および SQLExecute() 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。これはデフォルトです。
- 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 (DB2 CLI v6)

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を可視にす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 SQLExecDirect() および SQLExecute() 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を可視にす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、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。これはデフォルトです。
- 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。変更非可視カーソルは、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 (DB2 CLI v2)

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 = DB2 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 = 動的スクロール可能カーソルは、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 UDB 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 (DB2 CLI v8)

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ブール値属性。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。

SQL_ATTR_DEFERRED_PREPARE (DB2 CLI v5)

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 = 据え置き準備を使用不能にする。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) = 据え置き準備を使用可能にする。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使用不能にします。

注: 据え置き準備を使用可能にすると、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 DB2 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 (DB2 CLI v5)

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、CLI/ODBC ドライバーは、カーソルがクローズし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、ネットワーク要求を省略できるようになります。

  このオプションをオンにすると、小さい結果セットをたくさん使用するアプリケーションの処理速度が向上します。

次の場合は、EARLYCLOSE 機能を使用しません。
– ステートメントがブロッキングに適していない場合
– カーソル・タイプが SQL_CURSOR_FORWARD_ONLY 以外である場合

注: これは、IBM 定義の拡張機能です。

SQL_ATTR_ENABLE_AUTO_IPD (DB2 CLI v5)

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_FETCH_BOOKMARK_PTR (DB2 CLI v5)

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 *fFetchOrientation* で SQLFetchScroll() を呼び出すと、 DB2 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 (DB2 CLI v5)

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 (DB2 CLI v5)

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 (DB2 CLI v8)

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 は、ID を動的 SQL ステートメント・キャッシュに挿入されている任意のステートメントと関連付けます。

この属性は、DB2 UDB for z/OS バージョン 8 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 (DB2 CLI v8)

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。

SQL_ATTR_KEYSET_SIZE (DB2 CLI v5)

DB2 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_ATTR_LOAD_INFO (DB2 CLI v8)

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。

SQL_ATTR_LOAD_ROWS_COMMITTED_PTR (DB2 CLI v8)

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。

SQL_ATTR_LOAD_ROWS_DELETED_PTR (DB2 CLI v8)

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。

SQL_ATTR_LOAD_ROWS_LOADED_PTR (DB2 CLI v8)

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。

SQL_ATTR_LOAD_ROWS_READ_PTR (DB2 CLI v8)

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。

SQL_ATTR_LOAD_ROWS_REJECTED_PTR (DB2 CLI v8)

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。

SQL_ATTR_LOAD_ROWS_SKIPPED_PTR (DB2 CLI v8)

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。

SQL_ATTR_MAX_LENGTH (DB2 CLI v2)

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* 引き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可能な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() の呼び出しまたは取り出しは 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、DB2 CLI は文字またはバイナリー形式のすべての使用可能データを戻そうとします。

SQL_ATTR_MAX_ROWS (DB2 CLI v2)

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 (DB2 CLI v5)

カタログ関数のストリング引き数をどのように扱うかを判別する 32 ビット整数値。

- カタログ関数のストリング引き数 SQL_TRUE は、ID として扱われます。大文字小文字は区別されません。非区切りストリングの場合、DB2 CLI は後書きスペースがあれば除去し、ストリングは大文字変換されます。区切りストリングの場合、DB2 CLI は先行スペースまたは後書きスペースがあれば除去し、区切り文字の間を文字どおりに解釈します。これらの引き数の 1 つが NULL ポインターに設定されていると、関数は SQL_ERROR および SQLSTATE HY009 (NULL ポインターの無効な使用) を戻します。
- カタログ関数のストリング引き数 **SQL_FALSE** は、 ID として扱われません。大文字小文字の区別があります。引き数に応じて、ストリング検索パターンを含むことも含まないこともあります。

  これはデフォルト値です。

リストの値を取る SQLTables() の *TableType* 引き数は、この属性に影響されません。

SQL_ATTR_NOSCAN (DB2 CLI v2)

DB2 CLI が SQL をスキャンしてエスケープ文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、エスケープ文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文節を探しません。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。この選択を行うと、スキャンに関連したオーバーヘッド処理の一部が除かれます。

SQL_ATTR_OPTIMIZE_FOR_NROWS (DB2 CLI v6)

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″ 文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、この文節は追加されません。

OPTIMIZEFORNROWS DB2 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 (DB2 CLI v6)

32 ビット整数。

- 1 に設定すると、明示的な (ワイルドカードを使用しない) スキーマ名、明示的な表名、および列名の % (全列) が指定されている場合に、SQLColumns() の呼び出しが最適化されます。 DB2 CLI/ODBC ドライバーはこの呼び出しを最適化して、システム表がスキャンされないようにします。

  呼び出しが最適化されると、 SQLColumns() からの COLUMN_DEF 情報 (列のデフォルト・ストリングが入っている) は戻されず、 AS/400 NUMERIC 列のデータ・タイプが SQL_DECIMAL として戻されます。
- 0 (デフォルト値) に設定すると、情報は通常どおり戻されます。この設定は、アプリケーションが COLUMN_DEF 情報を必要とするときに使用してください。

OPTIMIZESQLCOLUMNS DB2 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_PARAM_BIND_OFFSET_PTR (DB2 CLI v5)

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。このフィールドが非 NULL の場合、DB2 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 (DB2 CLI v5)

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_PARAMETER_BIND_BY_COLUMN** (デフォルト値) に設定して、列ごとのバインドを選択します。

行ごと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 (DB2 CLI v5)

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、DB2 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 (DB2 CLI v5)

`SQLExecute()` または `SQLExecDirect()` 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: DB2 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、DB2 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 (DB2 CLI v2)

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_ATTR_PARAMS_PROCESSED_PTR (DB2 CLI v5)

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。

このフィールド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 (DB2 CLI v5)

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。

SQL_ATTR_PREFETCH (DB2 CLI v6)

サーバーが、現行ブロックの送信直後にデータの次のブロックをプリフェッチするかどうか (サーバーがサポートしている場合) を決める 32 ビット値。この値を指定すると、アプリケーションが現行ブロックを受信している間に、サーバーはデータの次のブロックを取得するようになります。

結果セット全体がデータの最初のブロックに収まる場合、またはカーソルが非ブロック・カーソル (たとえば、FOR UPDATE カーソルである場合、または結果に LOB データが入っている場合) である場合は、この設定は無効になります。

以下の値を指定することができます。

- SQL_PREFETCH_ON - サーバーがサポートしていれば、プリフェッチが行われます。
- SQL_PREFETCH_OFF - プリフェッチは行われません。これはデフォルトです。

SQL_ATTR_QUERY_OPTIMIZATION_LEVEL (DB2 CLI v6)

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。

サポートされている使用可能な値は -1 (デフォルト)、0、1、2、3、5、7、および 9 です。

SQL_ATTR_QUERY_TIMEOUT (DB2 CLI v2)

SQL ステートメント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、長時間の照会を終了するの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 DB2 CLI は無期限に待機することを意味します。 DB2 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。

SQL_ATTR_RETRIEVE_DATA (DB2 CLI v2)

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、行検索のオーバーヘッドを生じないで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 (DB2 CLI v8)

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ブール値属性。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。

SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI v5)

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、 DB2 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 (DB2 CLI v5)

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、DB2 CLI はポインターを据え置きし、据え置き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 (DB2 CLI v5)

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。引き数 **ValuePtr* に定義済み定数 SQL_BIND_BY_COLUMN を指定すると、列ごと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、行ごと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。

列ごとのバインドは、 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 (DB2 CLI v5)

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、DB2 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 (DB2 CLI v5)

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、DB2 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 (DB2 CLI v5)

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、DB2 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 (DB2 CLI v5)

これは、 SQLFetch() または SQLFetchScroll() への呼び出し後、取り出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 DB2 CLI により RowCountPtr 配列にマップされます。

SQL_ROWSET_SIZE (DB2 CLI v2)

現在、DB2 CLI アプリケーションは、 SQLExtendedFetch() ではなく SQLFetchScroll() を使用すべきです。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 (DB2 CLI v5)

このステートメント属性は、DB2 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。

シミュレートされた定位置の UPDATE および DELETE ステートメントの影響を、単一行だけにとどめるかどうかを指定する 32 ビット整数値。DB2 CLI は常に SQL_SC_UNIQUE の値を戻します。この値を変更すると、01S02 (オプション値を変更しました) を示す警告が出されます。

SQL_ATTR_STMTTXN_ISOLATION (DB2 CLI v2)

下記の SQL_ATTR_TXN_ISOLATION を参照してください。

SQL_ATTR_TXN_ISOLATION (DB2 CLI v2)

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 DB2 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 (DB2 CLI v5)

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、 DB2 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 (DB2 CLI v8)

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 ビットの整数。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 機能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。

関連概念:

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」の『パーティション・データベース環境におけるバッファー付き挿入』
- 「*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*」の『CLI アプリケーションのカーソル』
- 「*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*」の『CLI アプリケーションでの LOB ロケーター』
- 「*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*」の『マルチスレッド CLI アプリケーション』
- 「*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*」の『CLI アプリケーションでのユーザー定義タイプ (UDT) の使用法』

関連タスク:

- 「*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*」の『結果セットから返される行セットの指定』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『aslheapsz - 「アプリケーション・サポート層ヒープ・サイズ」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『rqrioblk - 「クライアント入出力ブロック・サイズ」構成パラメーター』
- 59 ページの『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』
- 93 ページの『SQLDescribeCol 関数 (CLI) - 列の属性のセットを戻す』

- 120 ページの『SQLExecute 関数 (CLI) - ステートメントの実行』
- 296 ページの『SQLRowCount 関数 (CLI) - 行カウントの取得』
- 330 ページの『SQLSetStmtAttr 関数 (CLI) - ステートメントに関連したオプションの設定』
- 「システム・モニター *ガイドおよびリファレンス*」の『appl_name アプリケーション名 : モニター・エレメント』
- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』
- 「*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*」の『CLI/ODBC 構成キーワード (カテゴリー別)』
- 「*管理 API リファレンス*」の『db2Load - ロード』
- 「*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*」の『ArrayInputChain CLI/ODBC 構成キーワード』

第 3 章 記述子 FieldIdentifier と初期設定値

この章では、記述子フィールドについて説明し、記述子のヘッダーとレコードのフィールドに初期設定する値のリストを示します。

記述子 FieldIdentifier 引き数の値 (CLI)

FieldIdentifier 引き数は、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 DB2 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、 DB2 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、DB2 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、 DB2 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、DB2 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 DB2 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。 DB2 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 DB2 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* 引き数のある SQLBindCol()、または NULL ポインターに設定された *Parameter ValuePtr* 引き数のある 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、 DB2 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。戻りコードが上記のいずれでもない場合は、 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、DB2 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 つまたは複数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 DB2 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、 DB2 CLI はいつでも SQL_DESC_TYPE フィールド内の値に含まれている DB2 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 152. 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、DB2 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、 DB2 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 動的引き数が指定されます。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、 DB2 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、DB2 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、DB2 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 引き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 DB2 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、 DB2 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 つまたは複数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、 DB2 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 つまたは複数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 DB2 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、動的引き数 (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き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、その動的引き数は NULL 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 1 つずつ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、 DB2 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、DB2 CLI は文字ストリングとバイナリー値が NULL 終了であるとみなします。(バイナリー値は NULL 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 DB2 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 DB2 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 文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 文節で使用できるが、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 文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 文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、DB2 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 DB2 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 153. 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。
SQL_DECIMAL、 SQL_NUMERIC	SQL_DESC_SCALE は 0 に設定されます。 SQL_DESC_PRECISION は、それぞれのデータ・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 または C タイプを指示することがあります。
SQL_INTERVAL	このデータ・タイプは DB2 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 (たとえば、“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_READ_ONLY。
- 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。

関連概念:

- 「*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*」の『CLI アプリケーションの記述子』
- 「*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*」の『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 179 ページの『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 184 ページの『SQLGetDescRec 関数 (CLI) - 記述子レコードの複数フィールド設定の取得』
- 309 ページの『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 319 ページの『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 414 ページの『記述子ヘッダーとレコード・フィールドの初期設定値 (CLI)』
- 「*SQL リファレンス 第 1 巻*」の『ユーザー定義タイプ』

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 154. ヘッダー・フィールドの初期設定

SQL_DESC_ALLOC_TYPE (SQLSMALLINT)			
R/W:	ARD: R APD: R IRD: R IPD: R	デフォルト値:	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)			
R/W:	ARD: R/W APD: R/W IRD: 未使用 IPD: 未使用	デフォルト値:	ARD: [a] APD: [a] IRD: 未使用 IPD: 未使用
SQL_DESC_ARRAY_STATUS_PTR (SQLUSMALLINT *)			
R/W:	ARD: R/W APD: R/W IRD: R/W IPD: R/W	デフォルト値:	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 *)			
R/W:	ARD: R/W APD: R/W IRD: 未使用 IPD: 未使用	デフォルト値:	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)			
R/W:	ARD: R/W APD: R/W IRD: 未使用 IPD: 未使用	デフォルト値:	ARD: SQL_BIND_BY_COLUMN APD: SQL_BIND_BY_COLUMN IRD: 未使用 IPD: 未使用
SQL_DESC_COUNT (SQLSMALLINT)			
R/W:	ARD: R/W APD: R/W IRD: R IPD: R/W	デフォルト値:	ARD: 0 APD: 0 IRD: D IPD: 0
SQL_DESC_ROWS_PROCESSED_PTR (SQLUINTEGER *)			

表 154. ヘッダー・フィールドの初期設定 (続き)

R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R/W		IRD: NULL ポインター
	IPD: R/W		IPD: NULL ポインター

a これらのフィールドは、IPD が DB2 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 155. レコード・フィールドの初期設定

SQL_DESC_AUTO_UNIQUE_VALUE (SQLINTEGER)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_BASE_COLUMN_NAME (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_BASE_TABLE_NAME (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_CASE_SENSITIVE (SQLINTEGER)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: R		IPD: D [a]
SQL_DESC_CATALOG_NAME (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_CONCISE_TYPE (SQLSMALLINT)			

表 155. レコード・フィールドの初期設定 (続き)

R/W:	ARD: R/W	デフォルト	ARD: SQL_C_DEFAULT
	APD: R/W	値:	APD: SQL_C_DEFAULT
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_DATA_PTR (SQLPOINTER)

R/W:	ARD: R/W	デフォルト	ARD: NULL ポインター
	APD: R/W	値:	APD: NULL ポインター
	IRD: 未使用		IRD: 未使用
	IPD: 未使用		IPD: 未使用 [b]

SQL_DESC_DATETIME_INTERVAL_CODE (SQLSMALLINT)

R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_DATETIME_INTERVAL_PRECISION (SQLINTEGER)

R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_DISPLAY_SIZE (SQLINTEGER)

R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用

SQL_DESC_FIXED_PREC_SCALE (SQLSMALLINT)

R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: R		IPD: D [a]

SQL_DESC_INDICATOR_PTR (SQLINTEGER *)

R/W:	ARD: R/W	デフォルト	ARD: NULL ポインター
	APD: R/W	値:	APD: NULL ポインター
	IRD: 未使用		IRD: 未使用
	IPD: 未使用		IPD: 未使用

SQL_DESC_LABEL (SQLCHAR *)

表 155. レコード・フィールドの初期設定 (続き)

R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_LENGTH (SQLUINTEGER)			
R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND
SQL_DESC_LITERAL_PREFIX (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_LITERAL_SUFFIX (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用
SQL_DESC_LOCAL_TYPE_NAME (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: R		IPD: D [a]
SQL_DESC_NAME (SQLCHAR *)			
R/W:	ARD: 未使用	デフォルト	ARD: ND
	APD: 未使用	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND
SQL_DESC_NULLABLE (SQLSMALLINT)			
R/W:	ARD: 未使用	デフォルト	ARD: ND
	APD: 未使用	値:	APD: ND
	IRD: R		IRD: N
	IPD: R		IPD: ND
SQL_DESC_NUM_PREC_RADIX (SQLINTEGER)			

表 155. レコード・フィールドの初期設定 (続き)

R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_OCTET_LENGTH (SQLINTEGER)

R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_OCTET_LENGTH_PTR (SQLINTEGER *)

R/W:	ARD: R/W	デフォルト	ARD: NULL ポインター
	APD: R/W	値:	APD: NULL ポインター
	IRD: 未使用		IRD: 未使用
	IPD: 未使用		IPD: 未使用

SQL_DESC_PARAMETER_TYPE (SQLSMALLINT)

R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IPD: 未使用		IPD: 未使用
	IRD: R/W		IRD: D=SQL_PARAM_INPUT

SQL_DESC_PRECISION (SQLSMALLINT)

R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_SCALE (SQLSMALLINT)

R/W:	ARD: R/W	デフォルト	ARD: ND
	APD: R/W	値:	APD: ND
	IRD: R		IRD: D
	IPD: R/W		IPD: ND

SQL_DESC_SCHEMA_NAME (SQLCHAR *)

R/W:	ARD: 未使用	デフォルト	ARD: 未使用
	APD: 未使用	値:	APD: 未使用
	IRD: R		IRD: D
	IPD: 未使用		IPD: 未使用

SQL_DESC_SEARCHABLE (SQLSMALLINT)

表 155. レコード・フィールドの初期設定 (続き)

	R/W:	ARD: 未使用	デフォルト値: ARD: 未使用
		APD: 未使用	APD: 未使用
		IRD: R	IRD: D
		IPD: 未使用	IPD: 未使用
SQL_DESC_TABLE_NAME (SQLCHAR *)			
	R/W:	ARD: 未使用	デフォルト値: ARD: 未使用
		APD: 未使用	APD: 未使用
		IRD: R	IRD: D
		IPD: 未使用	IPD: 未使用
SQL_DESC_TYPE (SQLSMALLINT)			
	R/W:	ARD: R/W	デフォルト値: ARD: SQL_C_DEFAULT
		APD: R/W	APD: SQL_C_DEFAULT
		IRD: R	IRD: D
		IPD: R/W	IPD: ND
SQL_DESC_TYPE_NAME (SQLCHAR *)			
	R/W:	ARD: 未使用	デフォルト値: ARD: 未使用
		APD: 未使用	APD: 未使用
		IRD: R	IRD: D
		IPD: R	IPD: D [a]
SQL_DESC_UNNAMED (SQLSMALLINT)			
	R/W:	ARD: 未使用	デフォルト値: ARD: ND
		APD: 未使用	APD: ND
		IRD: R	IRD: D
		IPD: R/W	IPD: ND
SQL_DESC_UNSIGNED (SQLSMALLINT)			
	R/W:	ARD: 未使用	デフォルト値: ARD: 未使用
		APD: 未使用	APD: 未使用
		IRD: R	IRD: D
		IPD: R	IPD: D [a]
SQL_DESC_UPDATABLE (SQLSMALLINT)			
	R/W:	ARD: 未使用	デフォルト値: ARD: 未使用
		APD: 未使用	APD: 未使用
		IRD: R	IRD: D
		IPD: 未使用	IPD: 未使用

a これらのフィールドは、IPD が DB2 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、 DB2 CLI は SQL_DESC_DATA_PTR が設定された値を返す必要はありません。

関連概念:

- 「*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*」の『CLI アプリケーションの記述子』
- 「*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*」の『CLI アプリケーションの記述子の整合性検査』

関連資料:

- 「*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*」の『CLI アプリケーション用の C データ・タイプ』
- 179 ページの『SQLGetDescField 関数 (CLI) - 記述子レコードの単一フィールド設定の取得』
- 309 ページの『SQLSetDescField 関数 (CLI) - 記述子レコードの単一フィールドの設定』
- 401 ページの『記述子 FieldIdentifier 引き数の値 (CLI)』

第 4 章 DiagIdentifier 引き数値

DiagIdentifier 引き数は検索する診断データ構造のフィールドを示します。この章では、可能なヘッダーとレコードのフィールドを説明します。

DiagIdentifier 引き数 (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* 引き数に組み込むことができます。記述子フィールドに定義されている唯一の診断ヘッダー・フィールドは、SQL_DIAG_NUMBER および SQL_DIAG_RETURNCODE です。

表 156. DiagIdentifier 引き数のヘッダー・フィールド

SQL_DIAG_CURSOR_ROW_COUNT (戻りタイプ SQLINTEGER)

このフィールドは、カーソルにある行のカウントを含みます。そのセマンティクスは、SQLGetInfo() 情報タイプに依存しています。

- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_STATIC_CURSOR_ATTRIBUTES2

これらは、それぞれのカーソル・タイプごとにどの行カウントが使用可能かを示しています (SQL_CA2_CRC_EXACT および SQL_CA2_CRC_APPROXIMATE ビットにおいて)。

このフィールドの内容が定義されるのは、ステートメント・ハンドルに対してのみであり、しかも SQLExecute()、SQLExecDirect()、SQLMoreResults() が呼び出された後でのみです。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* を指定して SQLGetDiagField() を呼び出すと、 SQL_ERROR が返されます。

SQL_DIAG_DYNAMIC_FUNCTION (戻りタイプ CHAR *)

これは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 (DB2 CLI がサポートしている値については、 426 ページの動的関数フィールドを参照してください)。このフィールドの内容が定義されるのは、ステートメント・ハンドルに対してのみであり、しかも SQLExecute() または SQLExecDirect() の呼び出しの完了後のみです。このフィールドの値は、SQLExecute() または SQLExecDirect() への呼び出し前には定義されていません。

表 156. DiagIdentifier 引き数のヘッダー・フィールド (続き)

SQL_DIAG_DYNAMIC_FUNCTION_CODE (戻りタイプ SQLINTEGER)
: これは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 (DB2 CLI がサポートしている値については、 426 ページの動的関数フィールドを参照してください)。このフィールドの内容が定義されるのは、ステートメント・ハンドルに対してのみであり、しかも SQLExecute() または SQLExecDirect() の呼び出しの完了後のみです。このフィールドの値は、SQLExecute()、SQLExecDirect()、または SQLMoreResults() への呼び出し前には定義されていません。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* を指定して SQLGetDiagField() を呼び出すと、 SQL_ERROR が返されます。このフィールドの値は、 SQLExecute() または SQLExecDirect() への呼び出し前には定義されていません。

SQL_DIAG_NUMBER (戻りタイプ SQLINTEGER)
: 指定されたハンドルで使用可能な状況レコードの数です。

SQL_DIAG_RELATIVE_COST_ESTIMATE (戻りタイプ SQLINTEGER)
: SQLPrepare() が正常に呼び出された場合、ステートメントを処理するのに必要なリソースの相対コスト見積もりが含まれます。据え置き準備が使用可能な場合、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。

SQL_DIAG_RETURNCODE (戻りタイプ 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により返される戻りコード。 *Handle* でまだ何の関数も呼び出されていなければ、SQL_DIAG_RETURNCODE には SQL_SUCCESS が返されます。

SQL_DIAG_ROW_COUNT (戻りタイプ SQLINTEGER)
: SQLExecute()、SQLExecDirect()、または SQLSetPos() 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数。これはカーソル指定が実行された後で定義されます。このフィールドの内容は、ステートメント・ハンドルでのみ定義されます。フィールド内のデータは、SQLRowCount() の *RowCountPtr* 引き数に返されます。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() で返される行カウントは、ステートメントが準備済みまたは割り当てられた状態に戻されるまでは同じ状態に保たれます。

レコード・フィールド

以下のレコード・フィールドを、*DiagIdentifier* 引き数に組み込むことができます。

表 157. DiagIdentifier 引き数のレコード・フィールド

SQL_DIAG_CLASS_ORIGIN (戻りタイプ CHAR *)
: このレコードにある 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。

  DB2 CLI は常に SQL_DIAG_CLASS_ORIGIN に空ストリングを返します。

SQL_DIAG_COLUMN_NUMBER (戻りタイプ SQLINTEGER)

表 157. DiagIdentifier 引き数のレコード・フィールド (続き)

SQL_DIAG_ROW_NUMBER フィールドで、行セットまたはパラメーター・セットにある行数が有効である場合、そのフィールドには結果セット内の列番号を示す値が入ります。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、その値は SQL_NO_COLUMN_NUMBER となります。 DB2 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。このフィールドの内容は、ステートメント・ハンドルでのみ定義されます。

SQL_DIAG_CONNECTION_NAME (戻りタイプ 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。

DB2 CLI は常に SQL_DIAG_CONNECTION_NAME に空ストリングを返します。

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_ROW_NUMBER (戻りタイプ SQLINTEGER)

このフィールドには、状況レコードに関連している、行セットにある行番号 (または、パラメーター・セットにあるパラメーター番号) が入ります。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。 DB2 CLI がこのレコードに関連している行番号を判別できなければ、フィールド値は SQL_ROW_NUMBER_UNKNOWN となります。このフィールドの内容は、ステートメント・ハンドルでのみ定義されます。

SQL_DIAG_SERVER_NAME (戻りタイプ 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。これは、*InfoType* に SQL_DATA_SOURCE_NAME を指定した `SQLGetInfo()` 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 (戻りタイプ CHAR *)

5 文字の SQLSTATE 診断コード。

SQL_DIAG_SUBCLASS_ORIGIN (戻りタイプ CHAR *)

SQL_DIAG_CLASS_ORIGIN と同じフォーマットおよび有効値のストリング。これは、SQLSTATE コードのサブクラスの定義部分を識別します。

DB2 CLI は常に SQL_DIAG_SUBCLASS_ORIGIN に空ストリングを返します。

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() 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。これは DB2 CLI が使用するリストであり、ODBC では、その他の値も指定します。

表 158. 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
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

関連概念:

- 「*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*」の『CLI アプリケーションでの据え置き準備』

関連資料:

- 188 ページの『SQLGetDiagField 関数 (CLI) - 診断データ・フィールドの取得』

第 5 章 データ・タイプ属性

この章では、DB2 CLI がサポートする SQL データ・タイプの以下の属性について説明します。

- 精度
- スケール
- 長さ
- 表示サイズ

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 159. 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。たとえば、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_NUMERIC	桁数の定義された最大数。たとえば、NUMERIC(10,3) と定義された列の精度は 10 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。たとえば、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_DATE [b]	10 (yyyy-mm-dd フォーマットの文字数)
SQL_TIME [b]	8 (hh:mm:ss フォーマットの文字数)

表 159. 精度 (続き)

fSqlType	精度
SQL_TIMESTAMP	TIMESTAMP データ・タイプで使用する、 "yyy-mm-dd hh:mm:ss[.fff[fff]]" フォーマットの文字数。たとえば、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_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	列またはパラメーターの定義された長さ。たとえば、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。たとえば、WCHAR(10) と定義された列の精度は 10 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() または SQLSetParam() の *cbParamDef* 引き数は無視されます。	

関連概念:

- 「*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*」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』

関連資料:

- 「*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*」の『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』

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、スケールは未定義であることに注意してください。小数点の右の桁数が固定されないからです。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 160. 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。たとえば、NUMERIC (10, 3) と定義された列のスケールは 3 です。

表 160. スケール (続き)

fSqlType	スケール
SQL_SMALLINT SQL_INTEGER SQL_BIGINT	0
SQL_REAL SQL_FLOAT SQL_DOUBLE	該当しません。
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	該当しません。
SQL_DATE SQL_TIME	該当しません。
SQL_TIMESTAMP	"yyyy-mm-dd hh:mm:ss[fff[fff]]" フォーマットの、小数点の右にある桁の数。たとえば、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。

関連概念:

- 「*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*」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』

関連資料:

- 「*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*」の『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』

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 NULL 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 161. 長さ

fSqlType	長さ
SQL_CHAR SQL_VARCHAR SQL_CLOB	列の定義された長さ。たとえば、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。たとえば、NUMERIC(10,3) と定義された列の長さは 12 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。たとえば、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_DATE SQL_TIME	6 (DATE_STRUCT または TIME_STRUCT 構造のサイズ)。
SQL_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。たとえば、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。たとえば、WCHAR(10) と定義された列の長さは 20 です。

関連概念:

- 「*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*」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』

関連資料:

- 「*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*」の『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』

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 162. 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。たとえば、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)。たとえば、NUMERIC(10,3) と定義された列の表示サイズは 12 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	列の定義された長さの 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)。たとえば、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_DATE	10 (yyyy-mm-dd フォーマットの日付)。
SQL_TIME	8 (hh:mm:ss フォーマットの時刻)。
SQL_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[fff[fff]]" フォーマットの文字数です。たとえば、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。たとえば、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。

関連概念:

- 「*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*」の『CLI アプリケーションにおけるデータ・タイプとデータ変換』

関連資料:

- 「*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*」の『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』

付録 A. DB2 Universal Database の技術情報の概要

DB2 ドキュメンテーションおよびヘルプ

DB2 の技術情報は、以下のツールや手段によって利用できます。

- DB2 インフォメーション・センター
  - トピック
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- ダウンロード可能な PDF ファイルおよび印刷された資料
  - ガイド
  - リファレンス・マニュアル
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ
  - SQL 状態ヘルプ
- インストールされているソース・コード
  - サンプル・プログラム

ibm.com において、オンラインでも付加的な DB2 Universal Database の技術情報を利用できます。その中には、技術情報、技術白書、およびレッドブック (Redbooks) が含まれています。 DB2 Information Management Library サイト (www.ibm.com/software/data/db2/udb/support.html) をご覧ください。

DB2 ドキュメンテーションの更新

IBM では、利用可能な DB2 インフォメーション・センターに対するドキュメンテーションのフィックスパックや、その他のドキュメンテーション更新情報を定期的に作成する場合があります。 http://publib.boulder.ibm.com/infocenter/db2help/ から DB2 インフォメーション・センターを利用する場合は、常にほとんど最新の情報を参照できます。 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た場合、更新された情報を表示するためには、更新情報をインストールする必要があります。ドキュメンテーション更新情報をインストールすると、新しい情報が利用可能になった場合に、 *DB2 インフォメーション・センター CD* からインストールした情報が更新されます。

インフォメーション・センターは、 PDF やハードコピー資料に比べて更新が頻繁です。 DB2 の最新の技術情報を入手するには、ドキュメンテーション更新情報が利用可能になった時点でそれをインストールするか、または www.ibm.com サイトの DB2 インフォメーション・センターを利用してください。

関連概念:

- 「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」の『CLI サンプル・プログラム』
- 「アプリケーション開発ガイド アプリケーションの構築および実行」の『Java サンプル・プログラム』
- 434 ページの『DB2 インフォメーション・センター』

関連タスク:

- 455 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 445 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 456 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 456 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 457 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 447 ページの『DB2 PDF 資料および印刷された資料』

DB2 インフォメーション・センター

DB2® インフォメーション・センターを使用すると、 DB2 Universal Database™、 DB2 Connect™、 DB2 Information Integrator および DB2 Query Patroller™ などの DB2 ファミリー製品を最大限に活用するのに必要なすべての情報にアクセスできます。また、DB2 インフォメーション・センターは、 DB2 の主な機能とコンポーネントに関する情報を提供します (レプリケーション、データウェアハウジング、および DB2 の種々の Extender など)。

Mozilla 1.0 以上または Microsoft® Internet Explorer 5.5 以上で表示する場合、 DB2 インフォメーション・センターには以下の機能があります。以下のいくつかの機能では、 JavaScript™ のサポートを使用可能にする必要があります:

柔軟なインストール・オプション
以下の中から、 ご使用の環境に最も適したオプションを使って DB2 資料を表示できます。

- 最新の資料を常に自動的に利用できるようにするには、 IBM® の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターからすべての資料に直接アクセスします。
- 更新処理を最小化し、イントラネット内のネットワーク・トラフィックだけに制限するには、イントラネット上の 1 つのサーバーに DB2 資料をインストールします。
- 柔軟性を改善し、ネットワーク接続への依存を軽減するには、個々のコンピューターに DB2 資料をインストールします。

検索 「**検索**」テキスト・フィールドに検索語を入力することにより、 DB2 インフォメーション・センターのすべてのトピックを検索できます。複数の語句を引用符で囲めば、完全一致を検索できます。また、ワイルドカード演算子 (*、?) とブール演算子 (AND、 NOT、OR) を使用して検索を絞り込むことができます。

タスク指向の目次

単一の目次の中から、DB2 資料のトピックを見付けることができます。目次は、主に実行するタスクの種類に従って編成されていますが、そのほかに製品概要、特定のゴール (目的) の情報、参照情報、索引、および用語集も含まれます。

- 製品概要では、DB2 ファミリーで使用可能な製品間の関係、そうした各製品で提供される機能、および各製品の最新リリース情報について説明されています。
- インストール、管理および開発などのゴール・カテゴリーには、タスクを迅速に完了し、そのための背景情報をよく理解できるようにするトピックが含まれています。
- 「参照」トピックでは、その対象に関する詳細な情報 (ステートメントとコマンドの構文、メッセージ・ヘルプ、構成パラメーターなど) が説明されています。

現在のトピックを目次に表示する

現在のトピックが目次のどの部分に該当するかを表示するには、目次フレーム内の**「リフレッシュ/現在のトピックの表示 (Refresh/Show Current Topic)」**ボタンをクリックするか、コンテンツ・フレーム内の**「目次に表示 (Show in Table of Contents)」**ボタンをクリックします。幾つかのファイルで関連トピックへの複数のリンクをたどった場合、または検索結果からトピックにアクセスした場合には、この機能が役立ちます。

索引 索引から、すべての資料にアクセスすることができます。索引では、用語が 50 音順に編成されています。

用語集 用語集を見れば、DB2 資料で使われているさまざまな用語の定義を調べることができます。用語集では、用語が 50 音順に編成されています。

組み込まれているローカライズ情報

DB2 インフォメーション・センターは、ブラウザーで設定された言語でトピックを表示します。設定された言語のトピックが利用できない場合、DB2 インフォメーション・センターにはそのトピックの英語版が表示されます。

iSeries™ 技術情報については、 IBM eServer™ iSeries Information Center (www.ibm.com/eserver/iseries/infocenter/) を参照してください。

関連概念:

- 436 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 445 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 446 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 439 ページの『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 (UNIX)』

- 442 ページの『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 (Windows)』

DB2 インフォメーション・センターのインストール・シナリオ

さまざまな作業環境にある人々が、それぞれの環境に応じた方法で DB2 製品資料にアクセスする必要があります。それで、DB2 製品資料にアクセスする方法には、IBM Web サイトからアクセスする方法、イントラネット・サーバーからアクセスする方法、そしてコンピューター上にインストールしてアクセスする方法の 3 種類があります。それら 3 種類のいずれにおいても、資料は DB2 インフォメーション・センターに含まれています。それは、ブラウザーで表示できるトピック単位の情報として設計された Web です。 DB2 製品は、デフォルトで IBM Web サイトから DB2 インフォメーション・センターにアクセスするようになっています。しかし、DB2 インフォメーション・センターをイントラネット・サーバーからアクセスしたり、自分のコンピューターでアクセスしたりしたい場合には、製品メディア・パックに含まれている DB2 インフォメーション・センター CD を使用することによって、 DB2 インフォメーション・センターをインストールする必要があります。以下に示す 3 つのシナリオを参考にすることにより、自分にとって、または自分の作業環境においてどの方法で DB2 インフォメーション・センターにアクセスするのが最善かを決定し、インストールに関して考慮しなければならない問題は何かを判別してください。

シナリオ: IBM Web サイトから DB2 インフォメーション・センターにアクセスする場合:

コリン氏は、ある教育機関の IT 関係の顧問をしています。彼は、データベース・テクノロジーや SQL を専門としており、北米全体の企業を対象として、 DB2 Universal Database Express Edition を使用したセミナーを開催しています。コリン氏のセミナーのある部分では、教材として DB2 のドキュメンテーションを使用します。たとえば、SQL の講座においてコリン氏は、データベース照会の基本的な構文や上級者向けの構文を教える手段として、 SQL に関する DB2 ドキュメンテーションを使用します。

コリン氏がセミナーを開く企業のほとんどにおいて、インターネットへのアクセスが可能です。それでコリン氏は、自分のモバイル・コンピューターに DB2 Universal Database Express Edition の最新版をインストールする際に、 IBM Web サイトにある DB2 インフォメーション・センターにアクセスするように設定することにします。それによりコリン氏は、セミナーの中で、最新の DB2 ドキュメンテーションにオンラインでアクセスできます。

しかし、コリン氏が旅行中はインターネットにアクセスできません。これは少し問題です。特に、セミナーの準備のために DB2 ドキュメンテーションにアクセスすることが必要になった場合に困ります。そのような状況に対処するため、コリン氏は、自分のモバイル・コンピューターにも DB2 インフォメーション・センターのコピーをインストールすることにします。

DB2 ドキュメンテーションのコピーがいつでも手元にあるので、コリン氏は柔軟に対応することができるようになりました。 **db2set** コマンドを使用して自分のモバイル・コンピューターのレジストリー変数の設定を容易に変更することにより、状

況に応じて、IBM Web サイトの DB2 インフォメーション・センターにアクセスしたり、自分のモバイル・コンピューター上にあるものを利用したりできます。

シナリオ: イントラネット・サーバー上の DB2 インフォメーション・センターにアクセスする場合:

エバ氏は、ある生命保険会社の主任データベース管理者です。管理者としての彼女の責任には、 DB2 Universal Database Enterprise Server Edition の最新バージョンを会社の UNIX データベース・サーバーにインストールおよび構成することが含まれます。彼女の会社では、最近従業員に対して、セキュリティー上の理由により業務中はインターネットへのアクセスができなくなることが通知されました。彼女の会社はネットワーク環境にあるため、エバ氏は DB2 インフォメーション・センターのコピーをイントラネット・サーバーにインストールすることにします。そのようにすれば、その会社のデータウェアハウスを定常的に使用する従業員の全員 (営業員、営業部長、および経営分析担当者) が DB2 インフォメーション・センターにアクセスできます。

DB2 インフォメーション・センターをイントラネット・サーバーにインストールする際にエバ氏は、 DB2 セットアップ・ウィザードから、 DB2 インフォメーション・センターがネットワーク上の他のコンピューターからの通信を受信するためのポートを指定するよう求められます。そこで彼女は、 DB2 インフォメーション・センターをインストールするイントラネット・サーバーのサービス名とポート番号を指定します。

次にエバ氏は、自分のデータベース・チームに対して、応答ファイルをしようして全従業員のコンピューターに DB2 Universal Database の最新バージョンをインストールするよう指示します。それにより各コンピューターは、イントラネット・サーバーのホスト名とポート番号を使用して DB2 インフォメーション・センターにアクセスするよう構成されます。

ところが、エバ氏のチームのデータベース管理者の一人であるミゲルが指示を聞き間違えて、イントラネット・サーバー上の DB2 インフォメーション・センターにアクセスするよう DB2 Universal Database を構成するのでなく、何人かの従業員のコンピューターに DB2 インフォメーション・センターのコピーをインストールしてしまいました。この状況を正すためエバ氏は、ミゲルに対して、 **db2set** コマンドを使用することにより、それらの各コンピューターの DB2 インフォメーション・センター・レジストリー変数 (ホスト名の変数は DB2_DOCHOST、ポート番号の変数は DB2_DOCPORT) を変更するように指示します。これで、ネットワーク上の該当するすべてのコンピューターは、 DB2 インフォメーション・センターにアクセスできるようになり、従業員は DB2 に関する質問の回答を DB2 ドキュメンテーションから調べることができるようになりました。

シナリオ: 自分のコンピューター上の DB2 インフォメーション・センターにアクセスする場合:

ツーチェン氏は、小さな町で工場を経営しています。そこにはインターネット・アクセスを提供する地元の ISP がありません。彼は、在庫管理、製品注文情報、銀行口座の情報、および経費を管理するために、DB2 Universal Database Personal

Edition を購入しました。それまで一度も DB2 製品を使用したことがなかったので、ツーチェン氏は、その使い方を調べるため DB2 製品資料を必要としています。

標準のインストール・オプションを使用して DB2 Universal Database Personal Edition を自分のコンピューターにインストールした後、ツーチェン氏は DB2 ドキュメンテーションを開こうとします。しかしブラウザーに、開こうとしたページが見つからないというエラー・メッセージが表示されます。ツーチェン氏は、「*DB2 Universal Database Personal Edition 概説およびインストール*」を見て、コンピューター上の DB2 ドキュメンテーションにアクセスするには DB2 インフォメーション・センターをインストールする必要があることを知ります。そこで、メディア・パックから、 *DB2 インフォメーション・センター CD* を取り出し、それをインストールします。

これでツーチェン氏は、オペレーティング・システムのアプリケーション・ランチャーから DB2 インフォメーション・センターを利用できるようになり、 DB2 製品を利用して作業効率を改善する方法について調べることができるようになりました。

DB2 ドキュメンテーションへのアクセス方法のサマリー:

各作業環境に応じて、 DB2 インフォメーション・センターに含まれる DB2 製品資料にアクセスするためにどのオプションが最善かを、次の表にまとめます。

インターネット・アクセス	イントラネット・アクセス	推奨
Yes	Yes	IBM Web サイトの DB2 インフォメーション・センターにアクセスするか、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・センターにアクセスする。
Yes	No	IBM Web サイトの DB2 インフォメーション・センターにアクセスする。
No	Yes	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・センターにアクセスする。
No	No	ローカル・コンピューター上の DB2 インフォメーション・センターにアクセスする。

関連概念:

- 434 ページの『DB2 インフォメーション・センター』

関連タスク:

- 445 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 439 ページの『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 (UNIX)』
- 442 ページの『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 (Windows)』

関連資料:

- 「コマンド・リファレンス」の『db2set - DB2 プロファイル・レジストリー・コマンド』

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 (UNIX)

DB2 製品資料にアクセスするには、IBM Web サイトからアクセスする方法、イントラネット・サーバーからアクセスする方法、そしてコンピューター上にインストールしてアクセスする方法の 3 種類の方法があります。 DB2 製品は、デフォルトで IBM Web サイトから DB2 ドキュメンテーションにアクセスするようになっています。イントラネット・サーバーまたは自分のコンピューターから DB2 ドキュメンテーションにアクセスするには、 *DB2 インフォメーション・センター CD* からドキュメンテーションをインストールする必要があります。 DB2 セットアップ・ウィザードを使用すると、インストール・システムのさまざまな設定値を定義し、 UNIX オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールすることができます。

前提条件:

ここでは、 UNIX コンピューターに DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の要件のリストを示します。

- **ハードウェア要件**

  以下のうちいずれか 1 つのプロセッサーが必要です。

  – PowerPC (AIX)
  – HP 9000 (HP-UX)
  – Intel 32 ビット (Linux)
  – Solaris UltraSPARC コンピューター (Solaris オペレーティング環境)

- **オペレーティング・システム要件**

  以下のうちいずれか 1 つのオペレーティング・システムが必要です。

  – IBM AIX 5.1 (PowerPC)
  – HP-UX 11i (HP 9000)
  – Redhat Linux 8.0 (Intel 32 ビット)
  – SuSE Linux 8.1 (Intel 32 ビット)
  – Sun Solaris バージョン 8 (Solaris オペレーティング環境 UltraSPARC コンピューター)

- **ソフトウェア要件**
  – 以下のブラウザーがサポートされています。
    - Mozilla バージョン 1.0 以上

- DB2 セットアップ・ウィザードは、グラフィック・インストーラーです。ご使用のコンピューターで DB2 セットアップ・ウィザードを実行するには、グラフィカル・ユーザー・インターフェースを表示できる X Window System ソフトウェ

アが必要です。 DB2 セットアップ・ウィザードを実行する前に、ディスプレイを正しくエクスポートしたことを確認してください。たとえば、以下のコマンドをコマンド・プロンプトに入力します。

```
export DISPLAY=9.26.163.144:0.
```

- **通信要件**
  - TCP/IP

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、

1. システムにログオンします。
2. DB2 インフォメーション・センター CD をシステムに挿入し、マウントします。
3. 次のコマンドを入力することによって、 CD がマウントされているディレクトリーに移動します。

   ```
   cd /cd
   ```

   /cd は CD のマウント・ポイントです。
4. **./db2setup** コマンドを入力して、 DB2 セットアップ・ウィザードを開始します。
5. **「IBM DB2 セットアップ・ランチパッド」** が表示されます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを表示するには、**「ヘルプ」**をクリックします。**「キャンセル (Cancel)」**を押せば、いつでもインストールを終了できます。
6. **「インストールしたい製品を選択します」**ウィンドウで、**「次へ」**をクリックします。
7. **「DB2 インフォメーション・センターの DB2 セットアップ・ウィザードへようこそ」**ウィンドウで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードにより、プログラムのセットアップ・プロセスが案内されます。
8. インストールを続行するには、ご使用条件に同意する必要があります。**「ご使用条件」**ウィンドウで、**「使用条件の条項に同意します」**を選択し、**「次へ」**をクリックします。
9. **「インストール・アクションの選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール先を選択します。後で応答ファイルを使用してこのコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをインストールする場合は、**「設定を応答ファイルに保管する」**を選択します。**「次へ (Next)」**をクリックします。
10. **「インストールする言語の選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール言語を選択します。**「次へ (Next)」**をクリックします。
11. **「DB2 インフォメーション・センター・ポートの指定」**で、 DB2 インフォメーション・センターが受け付ける通信について構成します。**「次へ」**をクリックして、インストールを続行します。

12. **「ファイルのコピーの開始」**ウィンドウで、それまでに選択したインストール・オプションを確認します。設定を確認または変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

DB2 インフォメーション・センターは、応答ファイルを使用してインストールすることもできます。

インストール・ログ db2setup.his、db2setup.log、および db2setup.err は、デフォルトでは /tmp ディレクトリーに入っています。ログ・ファイルの位置は指定可能です。

db2setup.log ファイルには、エラーを含めてすべての DB2 製品インストール情報が入れられます。 db2setup.his ファイルには、そのコンピューター上のすべての DB2 製品インストールが記録されます。 DB2 は db2setup.log ファイルを db2setup.his ファイルに付加します。 db2setup.err ファイルには、例外やトラップ情報など、Java から戻されたエラー出力が入れられます。

インストールが完了すると、使用している UNIX オペレーティング・システムに応じて、以下のいずれかのディレクトリーに 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ます。

- AIX: /usr/opt/db2_08_01
- HP-UX: /opt/IBM/db2/V8.1
- Linux: /opt/IBM/db2/V8.1
- Solaris オペレーティング環境 /opt/IBM/db2/V8.1

関連概念:

- 434 ページの『DB2 インフォメーション・センター』
- 436 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 「*インストールおよび構成 補足*」の『応答ファイルによる DB2 のインストール (UNIX)』
- 445 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 446 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 442 ページの『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 (Windows)』

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 (Windows)

DB2 製品資料にアクセスするには、IBM Web サイトからアクセスする方法、イントラネット・サーバーからアクセスする方法、そしてコンピューター上にインストールしてアクセスする方法の 3 種類の方法があります。 DB2 製品は、デフォルトで IBM Web サイトから DB2 ドキュメンテーションにアクセスするようになっています。イントラネット・サーバーまたは自分のコンピューターから DB2 ドキュメンテーションにアクセスするには、 *DB2 インフォメーション・センター CD* から DB2 ドキュメンテーションをインストールする必要があります。 DB2 セットアップ・ウィザードを使用すると、インストール・システムのさまざまな設定値を定義し、 Windows オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールすることができます。

前提条件:

ここでは、 Windows に DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の要件のリストを示します。

- **ハードウェア要件**

  以下のプロセッサーが必要です。

  – 32 ビット・コンピューター: Pentium または Pentium と互換の CPU。

- **オペレーティング・システム要件**

  以下のうちいずれか 1 つのオペレーティング・システムが必要です。

  – Windows 2000

  – Windows XP

- **ソフトウェア要件**

  – 次のブラウザーがサポートされています。

    - Mozilla 1.0 以上

    - Internet Explorer バージョン 5.5 または 6.0 (Windows XP の場合はバージョン 6.0)

- **通信要件**

  – TCP/IP

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、

1. DB2 インフォメーション・センターのインストールのために定義したアカウントで、システムにログオンします。
2. CD をドライブに挿入します。自動実行機能を使用可能にしている場合には、それによって IBM DB2 セットアップ・ランチパッドが起動されます。
3. DB2 セットアップ・ウィザードによりシステム言語が判別され、その言語用のセットアップ・プログラムが起動されます。セットアップ・プログラムを英語

以外の言語で実行したい場合や、セットアップ・プログラムが自動始動に失敗する場合には、 DB2 セットアップ・ウィザードを手動で開始することができます。

次のようにして、DB2 セットアップ・ウィザードを手動で開始します。

a. **「スタート」**をクリックし、**「ファイル名を指定して実行」**を選択します。
b. **「開く」**フィールドで、次のコマンドを入力します。

```
x:¥setup language
```

x: は CD のドライブ、 *language* はセットアップ・プログラムが実行されている言語です。

c. **「OK」**をクリックします。

4. **「IBM DB2 セットアップ・ランチパッド」** が表示されます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを表示するには、**「ヘルプ」**をクリックします。**「キャンセル (Cancel)」**を押せば、いつでもインストールを終了できます。
5. **「インストールしたい製品を選択します」**ウィンドウで、**「次へ」**をクリックします。
6. **「DB2 インフォメーション・センターの DB2 セットアップ・ウィザードへようこそ」**ウィンドウで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードにより、プログラムのセットアップ・プロセスが案内されます。
7. インストールを続行するには、ご使用条件に同意する必要があります。**「ご使用条件」**ウィンドウで、**「使用条件の条項に同意します」**を選択し、**「次へ」**をクリックします。
8. **「インストール・アクションの選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール先を選択します。後で応答ファイルを使用してこのコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをインストールする場合は、**「設定を応答ファイルに保管する」**を選択します。**「次へ (Next)」**をクリックします。
9. **「インストールする言語の選択」**ウィンドウで、 DB2 インフォメーション・センターのインストール言語を選択します。**「次へ (Next)」**をクリックします。
10. **「DB2 インフォメーション・センター・ポートの指定」**で、 DB2 インフォメーション・センターが受け付ける通信について構成します。**「次へ」**をクリックして、インストールを続行します。
11. **「ファイルのコピーの開始」**ウィンドウで、それまでに選択したインストール・オプションを確認します。設定を確認または変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

DB2 インフォメーション・センターは、応答ファイルを使用してインストールできます。また、**db2rspgn** コマンドを使用することによって、既存のインストール・システムに基づく応答ファイルを生成することもできます。

インストール時に検出されるエラーの詳細については、 db2.log と db2wi.log のファイルを参照してください。それらのファイルは、'My Documents'¥DB2LOG¥ ディレクトリーに入っています。 My Documents ディレクトリーのロケーションは、ご使用のコンピューターの設定によって異なります。

db2wi.log ファイルには、最後の DB2 インストール情報が入れられます。 db2.log には、DB2 製品インストールの履歴が入れられます。

関連概念:

- 434 ページの『DB2 インフォメーション・センター』
- 436 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 「*インストールおよび構成 補足*」の『応答ファイルによる DB2 製品のインストール (Windows)』
- 445 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 446 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 439 ページの『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 (UNIX)』

関連資料:

- 「コマンド・リファレンス」の『db2rspgn - 応答ファイル生成プログラム・コマンド』

DB2 インフォメーション・センターの呼び出し

DB2 インフォメーション・センターは、Linux、UNIX、および Windows オペレーティング・システム用の DB2 製品 (DB2 Universal Database、 DB2 Connect、DB2 Information Integrator、 DB2 Query Patroller など) を使用するために必要なすべての情報を提供します。

DB2 インフォメーション・センターは、以下の場所から呼び出すことができます。

- DB2 UDB クライアントまたはサーバーがインストールされているコンピューター
- 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イントラネット・サーバーまたはローカル・コンピューター
- IBM の Web サイト

前提条件:

DB2 インフォメーション・センターを呼び出すための要件は、以下のとおりです。

- *オプショナル*: 希望する言語でトピックを表示するようブラウザーを構成する
- *オプショナル*: 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・センターを使用するよう DB2 クライアントを構成する

手順:

DB2 UDB クライアントまたはサーバーがインストールされているコンピューターから DB2 インフォメーション・センターを呼び出すには、以下のようにします。

- (Windows オペレーティング・システムの)「スタート」メニューから: **「スタート」→「プログラム」→「IBM DB2」→「情報」→「インフォメーション・センター」**をクリックします。
- コマンド行プロンプトから:
  - Linux および UNIX オペレーティング・システムの場合、 **db2icdocs** コマンドを発行します。
  - Windows オペレーティング・システムの場合、 **db2icdocs.exe** コマンドを発行します。

イントラネット・サーバーまたはローカル・コンピュータ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 Web ブラウザーで開くには、以下のようにします。

- Web ページ http://<host-name>:<port-number>/ を開きます (<host-name> はホスト名、 <port-number> は DB2 インフォメーション・センターを利用可能なポート番号)。

IBM Web サイトにある DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ publib.boulder.ibm.com/infocenter/db2help/ を開きます。

関連概念:

- 434 ページの『DB2 インフォメーション・センター』

関連タスク:

- 446 ページの『DB2 インフォメーション・センターのトピックを特定の言語で表示する方法』
- 455 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 445 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 456 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 456 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 457 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール

http://publib.boulder.ibm.com/infocenter/db2help/ から利用できる DB2 インフォメーション・センターは、資料の新規追加または変更によって定期的に更新されます。さらに、更新された DB2 インフォメーション・センターをコンピューターまたはイントラネット・サーバーにダウンロードしてインストールできる場合もあります。DB2 インフォメーション・センターを更新しても、 DB2 クライアント製品またはサーバー製品は更新されません。

前提条件:

インターネットに接続されたコンピューターへのアクセスが必要です。

手順:

DB2 インフォメーション・センターの更新を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するには、以下のようにします。

1. IBM の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターを開きます。
2. 「DB2 インフォメーション・センターにようこそ」ページの見出し「サービスおよびサポート」の「ダウンロード」セクションで、**「DB2 資料」**リンクをクリックします。
3. 最新のドキュメンテーション・イメージのレベルと、インストール済みのドキュメンテーション・レベルを比較して、 DB2 インフォメーション・センターを更新する必要があるかどうかを確認します。「DB2 インフォメーション・センターにようこそ」ページに、インストール済みのドキュメンテーションのレベルがリストされます。
4. より新しいバージョンの DB2 インフォメーション・センターが存在する場合、ご使用のオペレーティング・システムに対応する最新の *DB2 インフォメーション・センター・イメージをダウンロードします。
5. 最新の *DB2 インフォメーション・センター・イメージをインストールするには、 Web ページの指示に従ってください。

関連概念:

- 436 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 439 ページの『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 (UNIX)』
- 442 ページの『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 (Windows)』

DB2 インフォメーション・センターのトピックを特定の言語で表示する方法

DB2 インフォメーション・センターを表示した場合、ブラウザーの設定値で指定された言語でトピックを表示することが試行されます。希望する言語に翻訳されていないトピックがある場合、そのトピックは英語で表示されます。

手順:

Internet Explorer ブラウザーにおいて、希望する言語でトピックを表示するには、

1. Internet Explorer で、**「ツール」→「インターネット オプション」→「言語...」**ボタンをクリックします。「言語の優先順位」ウィンドウが表示されます。
2. 言語リストの中で、希望する言語が最初の項目として指定されていることを確認してください。

- 新しい言語をリストに追加するには、**「追加...」**ボタンをクリックします。

  注: 言語を追加しても、その言語でトピックを表示するために必要なフォントがそのコンピューター上にあることが保証されるわけではありません。
- ある言語をリストの先頭に移動するには、その言語を選択してから、その言語が言語リストの先頭になるまで**「上へ」**ボタンをクリックします。

3. DB2 インフォメーション・センターが希望する言語で表示されるようにするため、ページをリフレッシュします。

Mozilla ブラウザーの場合に、希望する言語でトピックを表示するには、

1. Mozilla で、**「編集 (Edit)」→「設定 (Preferences)」→「言語 (Languages)」**ボタンを選択します。「設定 (Preferences)」ウィンドウに「言語 (Languages)」パネルが表示されます。
2. 言語リストの中で、希望する言語が最初の項目として指定されていることを確認してください。
   - 新しい言語をリストに追加するには、**「追加... (Add...)」**ボタンをクリックします。
   - ある言語をリストの先頭に移動するには、その言語を選択してから、その言語が言語リストの先頭になるまで**「上へ」**ボタンをクリックします。
3. DB2 インフォメーション・センターが希望する言語で表示されるようにするため、ページをリフレッシュします。

関連概念:

- 434 ページの『DB2 インフォメーション・センター』

DB2 PDF 資料および印刷された資料

以下の表は、正式な資料名、資料番号、および PDF ファイル名を示しています。ハードコピー版の資料を注文するには、正式な資料名を知っておく必要があります。 PDF ファイルを印刷するには、PDF ファイル名を知っておく必要があります。

DB2 資料は、以下のカテゴリーに分類されています。

- DB2 中核情報
- 管理情報
- アプリケーション開発情報
- ビジネス・インテリジェンス情報
- DB2 Connect 情報
- 入門情報
- チュートリアル情報
- オプショナル・コンポーネント情報
- リリース・ノート

以下の表は、DB2 ライブラリー内の各資料について、その資料のハードコピー版を注文したり、PDF 版を印刷または表示したりするのに必要な情報を示しています。

DB2 ライブラリー内の各資料に関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。

DB2 の基本情報

こうした資料の情報は、すべての DB2 ユーザーに基本的なもので、プログラマーおよびデータベース管理者にとって役立つ情報であるとともに、DB2 Connect、DB2 Warehouse Manager、または他の DB2 製品を使用するユーザーにとっても役立つ内容です。

表 163. DB2 の基本情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database コマンド・リファレンス*」	SC88-9140	db2n0j81
「*IBM DB2 Universal Database 用語集*」	資料番号なし	db2t0j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 1 巻*」	GC88-9152 (ハードコピーなし)	db2m1j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 2 巻*」	GC88-9153 (ハードコピーなし)	db2m2j81
「*IBM DB2 Universal Database 新機能*」	SC88-9158	db2q0j81

管理情報

これらの資料の情報は、DB2 データベース、データウェアハウス、およびフェデレーテッド・システムを効果的に設計し、インプリメントし、保守するために必要なトピックを扱っています。

表 164. 管理情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database 管理ガイド: プランニング*」	SC88-9135	db2d1j81
「*IBM DB2 Universal Database 管理ガイド: インプリメンテーション*」	SC88-9133	db2d2j81
「*IBM DB2 Universal Database 管理ガイド: パフォーマンス*」	SC88-9134	db2d3j81
「*IBM DB2 Universal Database 管理 API リファレンス*」	SC88-9136	db2b0j81
「*IBM DB2 Universal Database 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*」	SC88-9142	db2dmj81
「*IBM DB2 Universal Database 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*」	SC88-9143	db2haj81

表 164. 管理情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database データウェアハウス・センター 管理ガイド*」	SC88-9165	db2ddj81
「*IBM DB2 Universal Database SQL リファレンス 第 1 巻*」	SC88-9155	db2s1j81
「*IBM DB2 Universal Database SQL リファレンス 第 2 巻*」	SC88-9156	db2s2j81
「*IBM DB2 Universal Database システム・モニター ガイドおよびリファレンス*」	SC88-9157	db2f0j81

アプリケーション開発情報

これらの資料の情報は、 DB2 Universal Database (DB2 UDB) のアプリケーション開発者またはプログラマーが特に関心を持つ内容です。サポートされるさまざまなプログラミング・インターフェース (組み込み SQL、ODBC、JDBC、SQLJ、CLI など) を使用して DB2 UDB にアクセスするのに必要な資料とともに、サポートされる言語およびコンパイラーについても紹介されています。また、DB2 インフォメーション・センターをご使用の場合には、サンプル・プログラムのソース・コードの HTML バージョンにアクセスすることもできます。

表 165. アプリケーション開発情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database アプリケーション開発ガイド アプリケーションの構築および実行*」	SC88-9137	db2axj81
「*IBM DB2 Universal Database アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」	SC88-9138	db2a1j81
「*IBM DB2 Universal Database アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」	SC88-9139	db2a2j81
「*IBM DB2 Universal Database 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*」	SC88-9159	db2l1j81
「*IBM DB2 Universal Database 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*」	SC88-9160	db2l2j81
「*IBM DB2 Universal Database データウェアハウス・センター アプリケーション統合ガイド*」	SC88-9166	db2adj81

表 165. アプリケーション開発情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database XML Extender 管理およびプログラミングのガイド*」	SC88-9172	db2sxj81

ビジネス・インテリジェンス情報

これらの資料の情報は、さまざまなコンポーネントを使用して、 DB2 Universal Database のデータウェアハウジング機能および分析機能を拡張する方法を説明しています。

表 166. ビジネス・インテリジェンス情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Warehouse Manager Standard Edition インフォメーション・カタログ・センター 管理ガイド*」	SC88-9167	db2dij81
「*IBM DB2 Warehouse Manager Standard Edition インストール・ガイド*」	GC88-9164	db2idj81
「*IBM DB2 Warehouse Manager Standard Edition DB2 Warehouse Manager を使用時の ETI ソリューション・コンバージョン・プログラムの管理*」	SC88-9894	iwhe1mstx80

DB2 Connect 情報

このカテゴリーの情報は、 DB2 Connect Enterprise Edition または DB2 Connect Personal Edition を使用して、メインフレーム・サーバーおよびミッドレンジ・サーバー上のデータにアクセスする方法を説明しています。

表 167. DB2 Connect 情報

資料名	資料番号	PDF ファイル名
「*IBM コネクティビティー 補足*」	資料番号なし	db2h1j81
「*IBM DB2 Connect Enterprise Edition 概説およびインストール*」	GC88-9145	db2c6j81
「*IBM DB2 Connect Personal Edition 概説およびインストール*」	GC88-9146	db2c1j81
「*IBM DB2 Connect ユーザーズ・ガイド*」	SC88-9147	db2c0j81

入門情報

このカテゴリーの情報は、サーバー、クライアント、および他の DB2 製品をインストールして構成する場合に役立ちます。

表 168. 入門情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database DB2 クライアント機能 概説およびインストール*」	GC88-9144 (ハードコピーなし)	db2itj81
「*IBM DB2 Universal Database DB2 サーバー機能 概説およびインストール*」	GC88-9148	db2isj81
「*IBM DB2 Universal Database DB2 Personal Edition 概説およびインストール*」	GC88-9150	db2i1j81
「*IBM DB2 Universal Database インストールおよび構成 補足*」	GC88-9149 (ハードコピーなし)	db2iyj81
「*IBM DB2 Universal Database DB2 Data Links Manager 概説およびインストール*」	GC88-9141	db2z6j81

チュートリアル情報

チュートリアル情報は、DB2 機能を紹介し、さまざまなタスクを実行する方法を示します。

表 169. チュートリアル情報

資料名	資料番号	PDF ファイル名
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介*」	資料番号なし	db2tuj81
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド*」	資料番号なし	db2taj81
「*インフォメーション・カタログ・センター チュートリアル*」	資料番号なし	db2aij81
「*Video Central for e-business チュートリアル*」	資料番号なし	db2twj81
「*Visual Explain チュートリアル*」	資料番号なし	db2tvj81

オプショナル・コンポーネント情報

このカテゴリーの情報は、 DB2 のオプショナル・コンポーネントを使用する方法について説明しています。

表 170. オプショナル・コンポーネント情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Cube Views Guide and Reference*」	SC18–7298	db2aax81
「*IBM DB2 Query Patroller インストール、管理、使用法のガイド*」	GC88–9154	db2dwj81
「*IBM DB2 Spatial Extender and Geodetic Extender ユーザーズ・ガイドおよびリファレンス*」	SC88-9171	db2sbj81
「*IBM DB2 Universal Database Data Links Manager 管理ガイドおよびリファレンス*」	SC88-9169	db2z0x82
「*DB2 Net Search Extender 管理およびユーザーズ・ガイド*」 **注:** この資料の HTML 版は、HTML ドキュメンテーション CD からインストールされません。	SH88-8546	N/A

リリース・ノート

リリース・ノートは、ご使用の製品のリリースおよびフィックスパック・レベルに特有の追加情報を紹介します。また、リリース・ノートには、各リリース、アップデート、およびフィックスパックで組み込まれた資料上の更新の要約も含まれています。

表 171. リリース・ノート

資料名	資料番号	PDF ファイル名
「*DB2 リリース・ノート*」	「注」を参照。	「注」を参照。
「*DB2 インストール情報*」	製品 CD-ROM でのみ参照可能。	使用できません。

注: リリース・ノートは以下の形式で入手できます。

- XHTML およびテキスト形式 (製品 CD 内)
- PDF 形式 (PDF ドキュメンテーション CD 内)

さらに、リリース・ノートの中で、『*既知の問題と予備手段*』および『*リリース間の非互換性*』に関する部分は DB2 インフォメーション・センターにも表示されます。

UNIX ベースのプラットフォームでテキスト形式でリリース・ノートを確認するには、 Release.Notes ファイルを参照してください。このファイルは、 DB2DIR/Readme/%L ディレクトリーに収録されています。 %L はロケール名を表しています。 DB2DIR は以下になります。

- AIX オペレーティング・システムの場合: /usr/opt/db2_08_01
- その他のすべての UNIX ベースのオペレーティング・システムの場合: /opt/IBM/db2/V8.1

関連概念:

- 433 ページの『DB2 ドキュメンテーションおよびヘルプ』

関連タスク:

- 453 ページの『PDF ファイルからの DB2 資料の印刷方法』
- 454 ページの『DB2 の印刷資料の注文方法』
- 455 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』

PDF ファイルからの DB2 資料の印刷方法

DB2 PDF ドキュメンテーション CD に収録されている DB2 資料を印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷できます。

前提条件:

Adobe Acrobat Reader がインストールされていることを確認してください。 Adobe Acrobat Reader をインストールする必要がある場合、 Adobe Web サイト (www.adobe.com) から入手できます。

手順:

PDF ファイルから DB2 資料を印刷するには以下のようにします。

1. *DB2 PDF* ドキュメンテーション CD をドライブに挿入します。 UNIX オペレーティング・システムの場合、 DB2 PDF ドキュメンテーション CD をマウントします。 UNIX オペレーティング・システムで CD をマウントする方法については、「*概説およびインストール*」を参照してください。
2. index.htm を開きます。ブラウザー・ウィンドウにファイルが開きます。
3. 参照したい PDF のタイトルをクリックします。 Acrobat Reader で PDF が開きます。
4. **「ファイル」 → 「印刷」**を選択して、所要の資料の任意の部分を印刷します。

関連概念:

- 434 ページの『DB2 インフォメーション・センター』

関連タスク:

- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (AIX)』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『HP-UX 上での CD-ROM のマウント』

- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (Linux)』
- 454 ページの『DB2 の印刷資料の注文方法』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (Solaris)』

関連資料:

- 447 ページの『DB2 PDF 資料および印刷された資料』

DB2 の印刷資料の注文方法

ハードコピー版の資料を望む場合には、以下のいずれかの方法で注文できます。

印刷資料の注文方法:

一部の国または地域では、印刷された資料を注文することもできます。お客様がお住まいの国または地域でこのサービスが利用可能かどうかを確認するには、お住まいの国または地域の IBM Publications Web サイトをご覧ください。資料のご注文が可能な場合、以下のようにすることができます。

- 正規の IBM 製品販売業者または営業担当員に連絡してください。お客様がお住まいの地域の IBM 担当員の情報については、お手数ですが IBM の Web サイト (www.ibm.com/planetwide) の IBM Worldwide Directory of Contacts で確認してください。
- IBM Publications Center (http://www.ibm.com/shop/publications/order) にアクセスしてください。なお、IBM Publications Center から資料を注文できない国もあります。

DB2 製品がご利用可能になった時点で、印刷された資料は *DB2 PDF ドキュメンテーション CD* にある PDF 形式の資料と同じものです。さらに、 *DB2 インフォメーション・センター CD* に収録されている印刷された資料の内容もまた、これらと同じです。ただし、DB2 インフォメーション・センター CD には、 PDF 資料にない追加情報も含まれます (たとえば、SQL 管理作業や HTML サンプル)。 DB2 PDF ドキュメンテーション CD に収録されている資料の中には、ハードコピーとしてご注文できない資料もあります。

注: DB2 インフォメーション・センターは、 PDF またはハードコピー の資料よりも頻繁に更新されます。ドキュメンテーションの更新が入手可能になった時点でインストールするか、 DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) を参照して最新の情報を入手してください。

関連タスク:

- 453 ページの『PDF ファイルからの DB2 資料の印刷方法』

関連資料:

- 447 ページの『DB2 PDF 資料および印刷された資料』

DB2 ツールからコンテキスト・ヘルプを呼び出す

コンテキスト・ヘルプは、特定のウィンドウ、ノートブック、ウィザード、またはアドバイザーに関連したタスクまたはコントロールの情報を提供します。コンテキスト・ヘルプは、グラフィカル・ユーザー・インターフェースのある DB2 管理ツールおよび開発ツールから利用できます。コンテキスト・ヘルプには、以下の 2 種類があります。

- それぞれのウィンドウまたはノートブックにある**「ヘルプ」**ボタンからアクセス可能なヘルプ
- infopop (ポップアップ情報ウィンドウ)。これは、マウス・カーソルを特定のフィールドまたはコントロール上に置いたとき、またはウィンドウ、ノートブック、ウィザード、アドバイザー内でフィールドまたはコントロールを選択して F1 を押すと表示されます。

「ヘルプ」ボタンを押すと、概説、前提条件、およびタスク情報が表示されます。infopop は、それぞれのフィールドおよびコントロールについて説明します。

手順:

コンテキスト・ヘルプを呼び出すには、以下のようにします。

- ウィンドウおよびノートブックのヘルプを表示するには、いずれかの DB2 ツールを開始して、任意のウィンドウまたはノートブックを開きます。ウィンドウまたはノートブックの右下隅にある**「ヘルプ」**ボタンをクリックして、コンテキスト・ヘルプを呼び出します。

  また、それぞれの DB2 ツール・センターの上部にある**「ヘルプ」**メニュー項目からコンテキスト・ヘルプにアクセスすることもできます。

  ウィザードおよびアドバイザーでは、最初のページの「タスクの概要」リンクをクリックすると、コンテキスト・ヘルプを表示できます。
- ウィンドウまたはノートブック上の各コントロールの infopop ヘルプを表示するには、コントロールをクリックしてから、**F1** を押します。コントロールの詳細情報を示すポップアップ情報が、黄色いウィンドウに表示されます。

  注: フィールドまたはコントロールにマウス・カーソルを置いておくだけで infopops が表示されるようにするには、「ツール設定」ノートブックの**「文書 (Documentation)」**ページの**「infopops の自動表示」**チェック・ボックスを選択します。

  infopop に似た別のコンテキスト・ヘルプに、診断ポップアップ情報があります。これにはデータ入力規則が示されます。診断ポップアップ情報は、無効または不十分なデータが入力されたとき、紫色のウィンドウに表示されます。診断ポップアップ情報は、以下に関して表示されます。
  - 必須フィールド。
  - 日付フィールドのように、正確なフォーマットを必要とするデータのフィールド。

関連タスク:

- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 456 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』

- 456 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 457 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』
- 『DB2 UDB ヘルプの使用法: Common GUI help』

コマンド行プロセッサーからメッセージ・ヘルプを呼び出す

メッセージ・ヘルプは、メッセージが出された原因と、エラーへの応答として実行すべきアクションを説明します。

手順:

メッセージ・ヘルプを呼び出すには、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。

```
? XXXnnnnn
```

ここで、*XXXnnnnn* は有効なメッセージ ID を表します。

たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。

関連概念:

- 「*メッセージ・リファレンス 第 1 巻*」の『メッセージの概要』

関連資料:

- 「コマンド・リファレンス」の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーからコマンド・ヘルプを呼び出す

コマンド・ヘルプは、コマンド行プロセッサーでのコマンドの構文を説明します。

手順:

コマンド・ヘルプを呼び出すには、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。

```
? command
```

ここで *command* はキーワードまたはコマンド全体を表します。

たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、 `? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプだけが表示されます。

関連タスク:

- 455 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 456 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 457 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 「コマンド・リファレンス」の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーから SQL 状態ヘルプを呼び出す

DB2 Univerrsal Database は、 SQL ステートメントの結果の原因と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 状態ヘルプを呼び出すには、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。

たとえば、`? 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。

関連タスク:

- 444 ページの『DB2 インフォメーション・センターの呼び出し』
- 456 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 456 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』

DB2 チュートリアル

DB2® チュートリアルは、 DB2 Universal Database のさまざまな機能について学習するのを支援します。このチュートリアルでは、アプリケーションの開発、SQL 照会のパフォーマンス調整、データウェアハウスの処理、メタデータの管理、および DB2 を使用した Web サービスの開発の各分野で、段階的なレッスンが用意されています。

はじめに:

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

チュートリアルの中で、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、それぞれのチュートリアルを参照してください。

DB2 Universal Database チュートリアル:

以下に示すチュートリアルのタイトルをクリックすると、そのチュートリアルを表示できます。

ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介
データウェアハウス・センターを使用して簡単なデータウェアハウジング・タスクを実行します。

ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド
データウェアハウス・センターを使用して高度なデータウェアハウジング・タスクを実行します。

インフォメーション・カタログ・センター・チュートリアル
インフォメーション・カタログを作成および管理して、インフォメーション・カタログ・センターを使用してメタデータを配置し使用します。

Visual Explain チュートリアル
Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2® 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 ドキュメンテーション
トラブルシューティング情報は、DB2 インフォメーション・センター、および DB2 ライブラリーに含まれる PDF 資料の中でご利用いただけます。DB2 インフォメーション・センターで、 (ブラウザー・ウィンドウの左側の) ナビゲーション・ツリーの「サポートおよびトラブルシューティング (Support and troubleshooting)」ブランチを参照すると、 DB2 トラブルシューティング・ドキュメンテーションの詳細なリストが見つかります。

DB2 Technical Support の Web サイト
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検索したい場合は、 DB2 Technical Support の Web サイトを参照してください。Technical Support サイトには、最新の DB2 出版物、TechNotes、プログラム診断依頼書 (APAR)、フィックスパック、DB2 内部エラー・コードの最新リスト、その他のリソース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。

DB2 Technical Support の Web サイト
(http://www.ibm.com/software/data/db2/udb/winos2unix/support) にアクセスしてください。

DB2 Problem Determination Tutorial Series
DB2 製品で作業中に直面するかもしれない問題を素早く識別し、解決する方法に関する情報を見つけるには、 DB2 Problem Determination Tutorial Series の Web サイトを参照してください。あるチュートリアルでは、使用可能な DB2 問題判別機能およびツールを紹介し、それらをいつ使用すべきかを判断する助けを与えます。別のチュートリアルは、『データベース・エンジン問題判別 (Database Engine Problem Determination)』、『パフォーマンス問題判別 (Performance Problem Determination)』、『アプリケーション問題判別 (Application Problem Determination)』などの関連トピックを扱っています。

DB2 Technical Support
(http://www.ibm.com/software/data/support/pdm/db2tutorials.html) には、 DB2 問題判別チュートリアルがすべて揃っています。

関連概念:

- 434 ページの『DB2 インフォメーション・センター』
- 「*トラブルシューティング・ガイド*」の『Introduction to Problem Determination - DB2 テクニカル・サポートのチュートリアル』

アクセス支援

アクセス支援機能は、身体に障害のある (身体動作が制限されている、視力が弱いなど) ユーザーがソフトウェア製品を十分活用できるように支援します。 DB2® バージョン 8 製品に備わっている主なアクセス支援機能は、以下のとおりです。

- すべての DB2 機能は、マウスの代わりにキーボードを使ってナビゲーションできます。詳細については、『キーボードによる入力およびナビゲーション』を参照してください。
- DB2 ユーザー・インターフェースのフォント・サイズおよび色をカスタマイズすることができます。詳細については、 460 ページの『アクセスしやすい表示』を参照してください。
- DB2 製品は、Java™ Accessibility API を使用するアクセス支援アプリケーションをサポートします。詳細については、 460 ページの『支援テクノロジーとの互換性』を参照してください。
- DB2 資料は、アクセスしやすい形式で提供されています。詳細については、 460 ページの『アクセスしやすい資料』を参照してください。

キーボードによる入力およびナビゲーション

キーボード入力

キーボードだけを使用して DB2 ツールを操作できます。マウスを使って実行できる操作は、キーまたはキーの組み合わせによっても実行できます。標準のオペレーティング・システム・キー・ストロークを使用して、標準のオペレーティング・システム操作を実行できます。

キーまたはキーの組み合わせによって操作を実行する方法について、詳しくは キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・ナビゲーション

キーまたはキーの組み合わせを使用して、 DB2 ツールのユーザー・インターフェースをナビゲートできます。

キーまたはキーの組み合わせによって DB2 ツールをナビゲートする方法の詳細については、キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・フォーカス

UNIX® オペレーティング・システムでは、アクティブ・ウィンドウの中で、キー・ストロークによって操作できる領域が強調表示されます。

アクセスしやすい表示

DB2 ツールには、視力の弱いユーザー、その他の視力障害をもつユーザーのためにアクセシビリティーを向上させる機能が備わっています。これらのアクセシビリティー拡張機能には、フォント・プロパティーのカスタマイズを可能にする機能も含まれています。

フォントの設定

「ツール設定」ノートブックを使用して、メニューおよびダイアログ・ウィンドウに使用されるテキストの色、サイズ、およびフォントを選択できます。

フォント設定に関する詳細情報は、メニューおよびテキストのフォントを変更する: Common GUI help を参照してください。

色に依存しない

本製品のすべての機能を使用するために、ユーザーは必ずしも色を識別する必要はありません。

支援テクノロジーとの互換性

DB2 ツールのインターフェースは、Java Accessibility API をサポートします。これによって、スクリーン・リーダーその他の支援テクノロジーを DB2 製品で利用できるようになります。

アクセスしやすい資料

DB2 形式は、ほとんどの Web ブラウザーで表示可能な XHTML 1.0 形式で提供されています。 XHTML により、ご使用のブラウザーに設定されている表示設定に従って資料を表示できます。さらに、スクリーン・リーダーや他の支援テクノロジーを使用することもできます。

シンタックス・ダイアグラムはドット 10 進形式で提供されます。この形式は、スクリーン・リーダーを使用してオンライン・ドキュメンテーションにアクセスする場合にのみ使用できます。

関連概念:

• 460 ページの『ドット 10 進シンタックス・ダイアグラム』

ドット 10 進シンタックス・ダイアグラム

スクリーン・リーダーを使用してインフォメーション・センターを利用するユーザーのために、シンタックス・ダイアグラムがドット 10 進形式で提供されます。

ドット 10 進形式では、各シンタックス・エレメントは別々の行に書き込まれます。複数のシンタックス・エレメントが常に同時に存在する (または常に同時に不在の) 場合、単一のコンパウンド・シンタックス・エレメントとみなせるので同一行に表示できます。

各行は、ドット 10 進数で開始します。たとえば、3 または 3.1 ないしは 3.1.1 です。こうした数を適切に聞き取るには、スクリーン・リーダーが句読点を読み取るように設定されていることを確認してください。同じドット 10 進数を持つすべてのシンタックス・エレメント (たとえば、 3.1 という数値を持つすべてのシンタックス・エレメント) は、相互に排他的な代替エレメントです。 3.1 USERID および 3.1 SYSTEMID という行を聞き取る場合、シンタックスには両方ではなく USERID または SYSTEMID のどちらかが含まれることが分かります。

ドット 10 進レベルは、ネストのレベルを表示します。たとえば、ドット 10 進数 3 のシンタックス・エレメントの後に、一連のドット 10 進数 3.1 のシンタックス・エレメントが続きます。 3.1 の番号が付されたシンタックス・エレメントすべては、番号 3 の付されたシンタックス・エレメントに従属します。

シンタックス・エレメントに関する情報を追加するため、ドット 10 進数の次に特定のワードおよびシンボルが使用されます。 時折、こうしたワードおよびシンボルはエレメントの最初に表示される場合もあります。 簡単に識別するため、ワードやシンボルがシンタックス・エレメントの一部である場合には、円記号 (¥) 文字が先頭に付きます。 * シンボルはドット 10 進数の次に使用でき、シンタックス・エレメントが反復することを示します。 たとえば、ドット 10 進数 3 のシンタックス・エレメント *FILE は、3 ¥* FILE という形式になります。 3* FILE という形式は、シンタックス・エレメント FILE が反復されることを示します。 3* ¥* FILE という形式は、シンタックス・エレメント * FILE が反復されることを示します。

シンタックス・エレメントのストリングを分離するのに使用されるコンマなどの文字は、シンタックス内の分離する項目の直前に表示されます。 こうした文字は、それぞれの項目と同一行に表示するか、同じドット 10 進数を持つ関連する項目のある別の行に表示できます。 またその行には、シンタックス・エレメントに関する情報を提供する別のシンボルを表示することも可能です。 たとえば、複数の LASTRUN および DELETE シンタックス・エレメントを使用している場合には、5.1*、5.1 LASTRUN、および 5.1 DELETE という行は、エレメントをコンマで区切る必要があります。区切り文字が指定されないと、各シンタックス・エレメントを区切るのにブランクが使用されると想定されます。

シンタックス・エレメントの前に % シンボルが付く場合、他の箇所で定義されている参照であることを示します。 % シンボルの後のストリングは、リテラルではなくシンタックス・フラグメントの名前です。 たとえば、2.1 %OP1 という行は別のシンタックス・フラグメント OP1 を参照すべきことを意味します。

以下のワードおよびシンボルが、ドット 10 進数の次に使用されます。

- ? は、オプションのシンタックス・エレメントであることを表します。? シンボルが後に続くドット 10 進数は、対応するドット 10 進数のシンタックス・エレメント、および任意の従属のシンタックス・エレメントがオプションであることを示します。ドット 10 進数の付いたシンタックス・エレメントが 1 つしかない場合、 ? シンボルはそのシンタックス・エレメントと同じ行に表示されます (たとえば、5? NOTIFY)。ドット 10 進数の付いたシンタックス・エレメントが複数

ある場合、 ? シンボルだけで行に表示され、その後にオプションのシンタックス・エレメントが続きます。たとえば、「5 ?、5 NOTIFY、および 5 UPDATE」という行を聞き取る場合、シンタックス・エレメント NOTIFY および UPDATE がオプションである、つまりそのいずれかを選択でき、どちらも選択しないこともできることが分かります。 ? シンボルは、線路型ダイアグラムのバイパス線に相当します。

- ! は、デフォルトのシンタックス・エレメントであることを表します。! シンボルおよびシンタックス・エレメントが後に続くドット 10 進数は、そのシンタックス・エレメントが、同じドット 10 進数を共用するシンタックス・エレメントすべてのデフォルト・オプションであることを示します。同じドット 10 進数を共用するシンタックス・エレメントのうち 1 つだけに、 ! シンボルを指定できます。たとえば、「2? FILE、2.1! (KEEP)、および 2.1 (DELETE)」という行を聞き取る場合、 FILE キーワードのデフォルト・オプションは (KEEP) になります。この例では、FILE キーワードを含めてもオプションを指定しない場合には、デフォルト・オプション KEEP が適用されます。デフォルト・オプションは、次に高位のドット 10 進数にも適用されます。 この例の場合、FILE キーワードが省略されると、デフォルトの FILE(KEEP) が使用されます。しかし、「2? FILE、2.1、2.1.1! (KEEP)、および 2.1.1 (DELETE)」という行を聞き取る場合、デフォルト・オプション KEEP は次に高位のドット 10 進数 2.1 (関連キーワードを持っていない) にのみ適用され、 2? FILE には適用されません。キーワード FILE が省略されると、どれも使用されません。
- * は、0 回以上反復できるシンタックス・エレメントを示します。 * シンボルが後に続くドット 10 進数は、このシンタックス・エレメントが 0 回以上使用できること、つまりオプションであり、なおかつ反復できることを表します。たとえば、5.1* データ域という行を聞き取る場合、 1 つまたは複数のデータ域を含めるか、またはデータ域を全く含めないことが可能です。「3*、3 HOST、および 3 STATE」という行を聞き取る場合、 HOST、STATE をどちらか一方または両方同時に含めるか、どちらも含めないことができます。

  注:

  1. ドット 10 進数の後にアスタリスク (*) が付き、ドット 10 進数の付いた項目が 1 つしかない場合には、同じ項目を複数回反復できます。
  2. ドット 10 進数の後にアスタリスクが付き、ドット 10 進数の付いた項目が複数ある場合、リストから複数の項目を使用できますが、各項目を複数回使用することはできません。 前述の例では、HOST STATE と書くことはできますが、HOST HOST とは書けません。
  3. * シンボルは、線路型シンタックス・ダイアグラムのループバック線に相当します。
- + は、1 回以上含める必要のあるシンタックス・エレメントであることを示します。 + シンボルが後に続くドット 10 進数は、このシンタックス・エレメントを 1 回以上含める必要があること、つまり少なくとも 1 回は含める必要があり、反復できることを表します。たとえば、「6.1+ データ域」という行を聞き取る場合、データ域を少なくとも 1 回は含めなければなりません。「2+、2 HOST、および 2 STATE」という行を聞き取る場合には、 HOST、STATE、またはその両方を含める必要があります。 * シンボルと同様に、+ シンボルは、ドット 10 進

数の付いた項目が 1 つしかない場合に限り、その特定の項目のみを反復できます。 * シンボルと同様、 + シンボルは線路型シンタックス・ダイアグラムのループバック線に相当します。

関連概念:

- 459 ページの『アクセス支援』

関連タスク:

- 『目次』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『構文図の見方』

DB2 Universal Database 製品の共通基準認証

DB2 Universal Database は、 Common Criteria の評価検定レベル 4 (EAL4) で認証の評価を受けています。 Common Criteria の詳細については、以下の Common Criteria の Web サイトを参照してください。 http://niap.nist.gov/cc-scheme/

付録 B. 「DB2 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」の特記事項

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。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

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
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
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。

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。

本書には、X/Open Company Limited が著作権を持つテキストが含まれています。テキストは、許可を得て下記の資料から収録されています。

X/Open CAE Specification, March 1995,
Data Management: SQL Call Level Interface (CLI)
(ISBN: 1-85912-081-4, C451).

X/Open Preliminary Specification, March 1995,
Data Management: Structured Query Language (SQL), Version 2
(ISBN: 1-85912-093-8, P446).

本書には、Microsoft Corporation が著作権 (1992, 1993, 1994, 1997 年) を持つテキストが含まれています。テキストは、許可を得て下記の Microsoft の資料から収録されています。 *ODBC 2.0 Programmer's Reference and SDK Guide* ISBN 1-55615-658-8 および *ODBC 3.0 Software Development Kit and Programmer's Reference* ISBN 1-57231-516-4.

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

以下は、それぞれ各社の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。

Action Media、LANDesk、MMX、 Pentium および ProShare は Intel Corporation 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、Sun Microsystems, Inc. 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。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

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。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ラ行]

B

T

V

W

IBM と連絡をとる

技術上の問題がある場合は、お客様サポートにご連絡ください。

製品情報

DB2 Universal Database 製品に関する情報は、
http://www.ibm.com/software/data/db2/udb から入手できます。

このサイトには、技術ライブラリー、資料の注文方法、製品のダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースへのリンクに関する最新情報が掲載されています。

米国以外の国で IBM に連絡する方法については、 IBM Worldwide ページ (www.ibm.com/planetwide) にアクセスしてください。

Printed in Japan

SC88-9160-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12